Geomasking: Análisis y comparativa de algoritmos para perturbaciones de puntos.

Trabajo Final de Grado
Grado en Ingeniería Informática
Computación

Autor: Raúl Gómez Calero
Director: Rodrigo Ignacio Silveira

Enero de 2018

Como consecuencia inherente al avance tecnológico de las sociedades, es un hecho que los individuos han incrementado el número de dispositivos que poseen con capacidades de conectividad y/o posicionamiento geográfico.

En el presente proyecto se ha tratado el enmascaramiento geográfico, también conocido como geomasking, con el objetivo de preservar la privacidad de los usuarios que han proporcionado dichos datos. Se ha propuesto como objetivo, partiendo de una investigación previa teórica, detectar la existencia de una relación directa y/o indirecta entre el grado de enmascaramiento de los datos y los resultados obtenidos bajo métodos de análisis como Kernel Density Estimation (KDE).

Una vez especificadas las bases de inicio del proyecto y acotados sus objetivos, se ha realizado la implementación de uno de los algoritmos descritos en la investigación de la que se ha partido y se ha realizado un análisis de los resultados.

• • •

Com a conseqüència inherent a l'avanç tecnològic de les societats, és un fet que els individus han incrementat el nombre de dispositius que posseeixen amb capacitats de connectivitat i/o posicionament geogràfic.

En el present projecte s'ha tractat l'emmascarament geogràfic, també conegut com geomasking, amb l'objectiu de protegir la privacitat dels usuaris que han proporcionat aquestes dades. S'ha proposat com a objectiu, partint d'una recerca prèvia teòrica, detectar l'existència d'una relació directa i/o indirecta entre el grau d'emmascarament de les dades i els resultats obtinguts sota mètodes d'anàlisis com Kernel Density Estimation (KDE).

Una vegada especificades les bases d'inici del projecte i fitats els seus objectius, s'ha realitzat la implementació d'un dels algorismes descrits en la recerca de la qual s'ha partit i s'ha realitzat una anàlisi dels resultats.

• • •

As a consequence of the technological advance of societies, it is a fact that people has increased the number of devices they possess with connectivity and / or geographical positioning capabilities.

In the present project, geographic masking, also known as geomasking, has been treated with the aim of preserving the privacy of the users who have provided these data. The objective, starting from a theoretical previous investigation, is to detect the existence of a direct and / or indirect relationship between the degree of masking of the data and the results obtained under analysis methods such as Kernel Density Estimation (KDE).

Once the initial bases of the project have been specified and its objectives have been outlined, the implementation of one of the algorithms described in the original research and an analysis of the results has been carried out.

Índice

Índice de figuras

Índice de tablas

1. Contexto

1.1. Introducción

Dados los avances que se producen en una sociedad cada vez más tecnológica, los dispositivos que nos acompañan en nuestro día a día mejoran constantemente aumentando sus capacidades, bien en el aspecto de cómputo o por el número de periféricos que poseen. En éste segundo grupo y concretamente, en los datos de localización que nos proporcionan algunos de ellos, éste trabajo tiene el objetivo de centrarse.

Vivimos en una sociedad conectada, son ya pocas las personas o individuos capaces de entender su estilo de vida sin los dispositivos que nos acompañan diariamente y, consecuencia inherente al hecho descrito, es el creciente uso de dichos dispositivos con capacidad de posicionamiento que son capaces de obtener datos relacionados con la ubicación del usuario, siendo éstos reportados y/o almacenados de forma masiva.

Para proteger la privacidad de los datos geolocalizados, es posible aplicar un conjunto de técnicas, conocidas como geomasking o enmascaramiento geográfico, destinadas a la perturbación de datos de localización, con el objetivo de dificultar o imposibilitar la recuperación de las coordenadas originales. Existen, principalmente, dos formas de aplicarlo:

- **Agregación, también conocido por cloaking**:
  Método basado en dividir en regiones el mapa y agrupando las localizaciones por cada una de ellas, ocultando así los datos originales.

Figura 1: Ejemplo de utilizar el método de agregación sobre un conjunto de datos geográficos

- **Perturbación aleatoria, también conocida por dithering o jittering**:
  Método basado en modificar la ubicación original de cada uno de los datos a una ubicación aleatoria cercana en función de un radio r máximo de distancia definido.

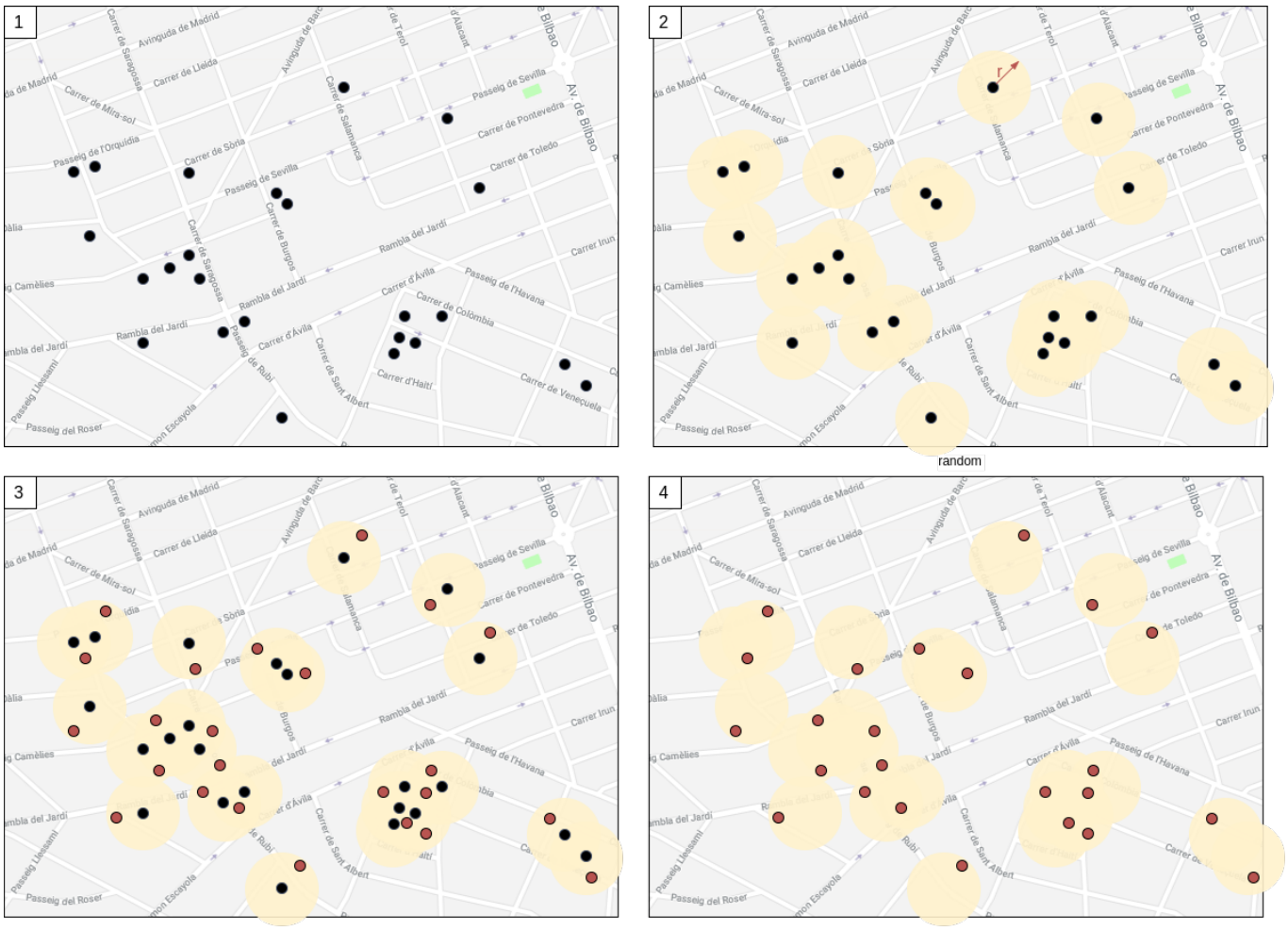


Figura 2: Ejemplo de utilizar el método de perturbación aleatoria con un radio r determinado

Para el análisis de los datos geográficos con métodos basados en el campo de la minería de datos, la perturbación aplicada produce variaciones en sus resultados.

Partiendo de una investigación teórica [1] centrada en el método de la perturbación aleatoria de datos geográficos y, basándose en la hipótesis que defiende la existencia de una relación directa entre los resultados obtenidos en métodos que son basados en círculos como Kernel Density Estimation KDE (explicado en la sección 4.1) y el número de puntos que contabiliza cada uno de los círculos utilizados, es posible controlar la desviación de resultados.
Se explicarán tres de los cuatro algoritmos propuestos en la investigación y se implementará uno de ellos con el objetivo de encontrar el radio máximo de perturbación r que permite una desviación δ máxima respecto al número de puntos por círculos.

En este trabajo se trata el enmascaramiento de los puntos geográficos con el objetivo de preservar la privacidad de los usuarios que los han proporcionado y realizar un análisis respecto los parámetros utilizados y sus resultados para observar si existen relaciones directas entre ellos.

1.2. Actores implicados

A continuación se detallan los actores implicados en el proyecto, es decir, las personas o individuos que pueden estar interesados y/o participan en el proyecto.

Desarrollador, analista y tester

Las tareas de programador, analista y tester serán realizadas por mí, dado que soy el único que desarrolla el proyecto.

Tutor del proyecto

El tutor de este proyecto es Rodrigo I. Silveira, su papel es el de supervisar que el proyecto cumpla el calendario estipulado así como alcanzar los objetivos definidos. Por otro lado, dado su conocimiento en el área de trabajo y que ha sido uno de los autores del artículo en el que se basa el proyecto, puede guiar al desarrollador en el curso de la realización del proyecto.

Investigadores

El proyecto se basa en la aplicación e implementación de uno de los algoritmos propuestos en un artículo de investigación publicado recientemente con el objetivo de analizar aspectos y relaciones entre datos de entrada y salida. De hecho, el análisis y test de sus resultados son la continuación del artículo y pueden ser de interés científico.

1.3. Formulación del problema

Cómo explorar los datos de localización para abstraer la información útil es una cuestión de vital importancia en la minería de datos y más concretamente, en la minería de datos espaciales.

Dado que gracias a los datos recibidos se pueden obtener y descubrir clústers espaciales interesantes para ciertos aspectos de investigación, es cada vez mayor su interés por parte de organizaciones y/o empresas que ven en ellos posibilidades de beneficio.

Sin olvidar todo lo que proporcionan, es necesario introducir la cuestión de la confidencialidad, pues los resultados generados pueden revelar información personal del domicilio, la dirección de trabajo u otra información confidencial perteneciente a los usuarios. Por este motivo, en este proyecto se ha querido tratar el concepto de geomasking, introducido en la sección 1.1.

Los objetivos de la minería de datos y la protección de la privacidad a menudo se encuentran bajo conflicto dado que, por un lado, el objetivo en última instancia es encontrar resultados significativos del conjunto de datos obtenidos y, en el otro, proteger la privacidad, hecho que requiere modificar o alterar los datos originales.

Como ejemplo a destacar, el área de la salud es de las más extendidas con técnicas de geomasking, pues los investigadores de la salud pública que usan datos de individuos deben aplicar ciertos algoritmos de geomasking para cumplir con la ética y los requisitos legales que se les exige [2].

Los problemas geométricos estudiados en este proyecto parten de la base del artículo [1], centrado en la posibilidad de encontrar relaciones entre las desviaciones de puntos en métodos basados en círculos, específicamente en el uso de KDE, una herramienta estándar utilizada en la búsqueda de clústers, y los resultados obtenidos.

El objetivo de este trabajo es centrarse en la perturbación de datos espaciales de forma aleatoria, puesto que partiendo de la hipótesis de [1], del umbral de perturbación máximo encontrado depende la distorsión de los patrones obtenidos como resultado para métodos basados en círculos en la búsqueda de clústers.

Por ende y bajo las premisas enunciadas anteriormente, el principal objetivo del proyecto será estudiar y analizar si limitar en un cierto porcentaje los puntos que pueden cambiar de círculo su pertenencia al perturbar, se traduce en una variación de los resultados de KDE proporcional a dicho porcentaje.

1.4. Objetivos del proyecto

Definidos los problemas a resolver, se detallan a continuación los objetivos:

- Realizar una búsqueda, estudio y análisis de artículos e investigaciones para identificar los diferentes escenarios de aplicación del método KDE, que se basa en generar sus resultados en los datos de puntos que son interiores a los círculos determinados por éstos.
- Para cada uno de los artículos, identificar los parámetros usados para los círculos, obtener si es posible los datos representativos originales o similares utilizados y considerar diferentes tipos, tamaños y estructuras con el objetivo de obtener un conjunto variado de datos espaciales.
- Analizar las relaciones de parámetros punto/círculo propuestos en [1] para cada dataset y obtener un análisis de su complejidad, ya que estos parámetros determinan el coste computacional de los métodos propuestos en [1].
- Implementar un algoritmo definido en [1] y posteriormente ejecutarlo con cada dataset.
- Comparar la relación de los parámetros analizados en los diferentes datasets y sus implicaciones en aspectos de complejidad y resultados.

2. Estado del arte

Con el fin de describir el estado del arte de este proyecto se procederá a analizar el algoritmo que generará los resultados de clústers (KDE), que serán procesados para analizar las diferencias obtenidas.

Por otro lado servirá para realizar una explicación y facilitar la comprensión de los conceptos, funcionamiento y comportamiento de los algoritmos.

2.1. Kernel Density Estimation, KDE

Kernel Density Estimation es una técnica utilizada para estimar una función de densidad que ayuda a identificar regiones de mayores y menores probabilidades para valores de una variable aleatoria.

Sea X una cantidad cualquiera aleatoria que tenga una probabilidad f, la especificación de la función de f proporciona una descripción de la distribución de X y permite obtener las probabilidades de X asociadas.

Sean n los puntos proporcionados en un dataset D cualquiera, supongamos que se parte de tales puntos y que éstos son una muestra de una función de densidad de probabilidad desconocida.
La estimación de la densidad, tal como la analiza KDE, es la construcción de una estimación de la función de densidad a partir de los datos observados.

Sea x un punto cualquiera del plano a estimar su densidad, h el ancho de banda y p_1, p_2 y p_3 puntos en el plano, la estimación de la densidad viene definida por un estimador de Kernel, definido a continuación.

2.1.1. El estimador de Kernel

Tal como se explica en [3], se define como función de peso una función de kernel K que satisface la siguiente condición:

$$\int_{-\infty}^{+\infty} K(x)dx = 1$$

Por lo general, se utiliza K como una función de densidad de probabilidad simétrica como puede ser la distribución normal [23].

El estimador con núcleo K queda definido por:

$$f(x) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - X_i}{h})$$

donde x_1, x_2, ..., x_n son las variables aleatorias de la distribución desconocida, n el número de muestras, K el estimador del Kernel y h es el ancho de banda, también conocido como parámetro de suavizado o bandwidth.

Para más dimensiones, dos en el presente proyecto, se generaliza la estimación de densidad así como cada uno de sus parámetros como se puede comprobar a continuación. Para datos en dos dimensiones, a la hora de evaluar las probabilidades de los datos, existen dos métodos de análisis descritos a continuación.

2.2. Análisis de datos en KDE

Análisis por Grid:

1. El plano sobre el que se representan los puntos se divide en una matriz de celdas de tamaño sz determinado.
2. Se trazan círculos de radio fijo h, con radio igual al ancho de banda, alrededor de cada centro de celda.
3. Se aplica el estimador con núcleo K definido anteriormente en cada uno de los círculos generados y se asigna, dicho valor, con la estimación de la densidad como densidad de la celda a la que pertenecen.

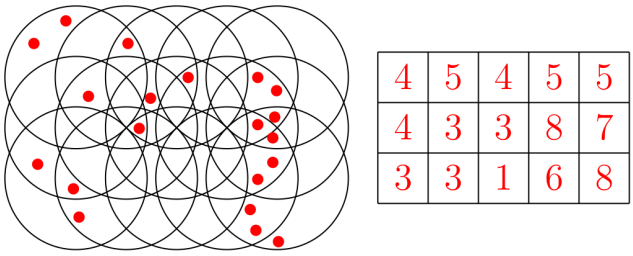


Figura 3: Ejemplo simplificado del método grid extraída de [1]

Análisis por Kernel:

1. Se trazan tantos círculos de radio fijo h, con radio igual al ancho de banda, como puntos tenga el dataset, fijando como centro de cada uno el punto al que le corresponda.

2. Se aplica el estimador con núcleo K definido anteriormente en cada uno de los círculos generados y se asigna, dicho valor, con la estimación de la densidad como la densidad del círculo al que pertenecen.

Estudios anteriores realizados tratando de entender el efecto de la perturbación en los patrones obtenidos de KDE, tratan dicho algoritmo como una caja negra que no puede ser modificada y únicamente varían empíricamente o estudian la combinación de valores de umbral de perturbación y bandwidth que genera buenos resultados [4, 5, 6].

En [1] se propone abordar el problema desde un punto de vista totalmente opuesto. Partiendo del criterio principal que define KDE, que es la relación de pertenencia de puntos y círculos (aún cuando el peso de cada muestra varíe al modificarse su ubicación), se propone buscar el umbral de perturbación máximo que preserva todas las pertenencias de los puntos en sus respectivos círculos.

En una segunda fase, se busca el umbral de perturbación máximo (definido por el usuario) que garantice cambios mínimos en las relaciones.

El principal objetivo de este trabajo es analizar si la variación de pertenencia de puntos a círculos está directamente relacionada con la variación de los resultados de KDE.

2.3. Conceptos básicos

Siguiendo los conceptos introducidos, se define:

- Para un punto p_j y círculo C_j con centro c_j y radio r_j, la distancia d_{ij} entre p_i y c_j es

$$d_{ij} = ||p_i - c_j||$$

- Si el punto esta dentro de C_j, el umbral de perturbación máximo para p_i y c_j en el problema 1 será de $r_j - d_{ij}$
- Si el umbral es inferior a ese valor, p_i se garantiza que esta fuera de C_j y el umbral de perturbación máximo es $d_{ij} - r_j$
- Se ha de tener en cuenta que $|d_{ij} - r_j|$ equivale a la distancia de p al punto más cercano de C_j.

2.4. Algoritmos para encontrar el umbral de perturbación máxima

A continuación, se prosigue a la descripción de algoritmos que han sido teorizados en [1] que pueden encontrar el valor umbral de perturbación máxima de forma automática.

2.4.1. Algoritmo de fuerza bruta

Primera fase del algoritmo: Buscar el umbral de perturbación r máximo que preserva todas las pertenencias de los puntos a sus respectivos círculos

El umbral máximo de perturbación r es el valor mínimo de $|d_{ij} - r_j|$, donde d_{ij} es la distancia mínima entre un punto i y un círculo j. Por tanto, la solución por fuerza bruta requiere la computación de todas las distancias entre todos los puntos con todos los círculos para obtener dicha distancia mínima, requiriendo un coste temporal de $\Theta(nm)$.

Segunda fase del algoritmo: Una vez encontrada la solución para la primera fase, modificar la búsqueda del umbral de perturbación máximo r tal que modifique como máximo en un porcentaje δ, la pertenencia original de los puntos a los círculos.

Para encontrar el umbral máximo de perturbación r tal que el número de puntos en cada círculo sea modificado como máximo en un porcentaje δ, se ha de computar todos los puntos que recaen dentro de cada círculo. Se define un umbral t_j para el cuál se calculan cuántos puntos pueden moverse dentro o fuera de C_j, buscando los puntos más cercanos a C_j en el interior y exterior.

Pudiéndose encontrar en tiempo $O(n)$, para todos los círculos toma una complejidad temporal de $O(nm)$.
Teorema 1 de [1]: Los problemas 1 y 2 pueden ser resueltos en tiempo $O(nm)$.

2.4.2. Basado en Voronoi

Primera fase del algoritmo:

Siendo equivalente a encontrar la distancia mínima entre puntos y círculos, es posible utilizar diagramas de Voronoi para encontrar ese valor [7, 8].

Se realiza un diagrama de Voronoi de los círculos C, que subdivide el plano en regiones en las que el límite de círculo más cercano es el mismo y puede utilizarse para encontrar la distancia mínima entre puntos y círculos. La complejidad del diagrama de Voronoi para m círculos es $O(p + m)$, y se puede calcular en tiempo $O((p + m) \log m)$.

Una vez calculado el diagrama de Voronoi del conjunto de círculos C, podemos encontrar el círculo más cercano a cada punto usando estructuras de datos de localización y permitiría consultas de ubicación en tiempo $O(\log(p + m))$. [1]

Segunda fase del algoritmo:

Para este problema, el enfoque Voronoi no funciona directamente, ya que requiere de igual forma que en fuerza bruta, saber cuántos puntos pertenecen a cada círculo y, por tanto en este apartado, Voronoi supone un cuello de botella.

2.4.3. Problema de conteo

Considerando la pregunta de contar el número de n puntos que recaen en cada uno de los m círculos, se puede encontrar como solución que la transformación de puntos y círculos de $\mathbb{R}^2$ a $\mathbb{R}^3$ [10] usando un mapa de elevación estándar, es decir, de (x, y) a $(x, y, x^2 + y^2)$) puede optimizar el tiempo de ejecución gracias a las estructuras de datos.

Agarwal [9] muestra que un conjunto de n puntos puede ser procesado en una estructura de datos de tamaño s tal que se puede resolver la consulta en tiempo de $O(\frac{n^{4/3}}{s^{2/3}} log(s/n))$, transformando el total de tiempo para m consultas en $O(n^{4/5} m^{3/5} log \frac{m^3}{n})$, que es una mejora clara respecto a la fuerza bruta cuadrática. [1]

Por tanto, es posible aproximar el cálculo a cualquier precisión deseada una búsqueda binaria estándar.

3. Definición del alcance

El alcance del proyecto comprende tanto el análisis como el desarrollo de una solución posible, que sigue con los objetivos establecidos con anterioridad y ajustado a ellos:

De forma exhaustiva, se han realizado los siguientes pasos:

1. Búsqueda exhaustiva de librerías que puedan crear así como visualizar el KDE de forma que simplifiquen la tarea de ese aspecto del proyecto.
2. Realizar una búsqueda, estudio y análisis de artículos e investigaciones para identificar los diferentes escenarios de aplicación de métodos KDE, que se basan al generar sus resultados en los datos de puntos que son interiores a los círculos determinados por éstos.
3. Para cada uno de los artículos, identificar los parámetros usados para los círculos, obtener si es posible los datos representativos originales o similares utilizados y considerar diferentes tipos, tamaños y estructuras con el objetivo de obtener un conjunto variado de datos espaciales.
4. Analizar las relaciones de parámetros punto/círculo propuestos en [1] para cada dataset y obtener un análisis de su complejidad.
5. Implementar el algoritmo de fuerza bruta definido y posteriormente ejecutarlo con cada dataset.
6. Comparar la relación de los parámetros analizados en los diferentes datasets y sus implicaciones en aspectos de complejidad y resultados.
7. Extraer conclusiones

3.1. Visualización de datos y resultados

Aún cuando el objetivo del trabajo no es generar un visor de datos para KDE ni implementar versiones de KDE, es necesario disponer de estos elementos con el objetivo de realizar comprobaciones y tests. Así pues, se han utilizado las librerías que ya implementan estos aspectos de la parte de programación y visualización.

3.2. Implementación de los algoritmos

Para realizar la parte de computación del algoritmo de búsqueda de la distancia máxima de perturbación y los análisis que se aplican a los datasets y resultados (explicados en profundidad en la Sección 5), se ha determinado que la mejor opción es programar en lenguaje C++ en conjunción con la librería CGAL [11], librería centrada en la geometría computacional que ya implementa métodos geométricos con precisión, control de errores y manejo de estructuras complejas.

3.3. Conclusiones del alcance definido

Se ha implementado el algoritmo de Brute Force definido para determinar los umbrales máximos de perturbación descritos con anterioridad, utilizando la librería CGAL y el lenguaje C++.

En la parte de visualización de datos y de generación de clústers bajo el algoritmo KDE, se ha decidido utilizar Matlab.

4. Búsqueda de parámetros y datasets utilizados

4.1. Parámetros para KDE

Siguiendo los objetivos definidos en el alcance del proyecto (Sección 3) y tal como se ha explicado en el apartado 2.1, Kernel Density Estimation es una técnica utilizada para estimar una función de densidad, por lo general simétrica, que crea una estimación de densidad a partir de los datos observados.

Los parámetros relevantes para KDE (Ver Figura 4) son:

- Tamaño de celda, *cs* (Únicamente para el método Grid)
- Ancho de banda o bandwidth, *h*

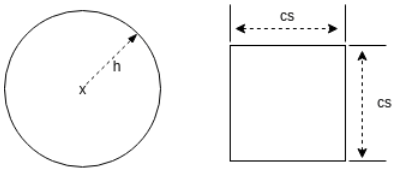


Figura 4: Visualización de los parámetros en KDE

Sean n los puntos proporcionados por un dataset cualquiera D, se pueden evaluar los puntos sobre un plano con dos métodos diferenciados (Ver sección 2.2):

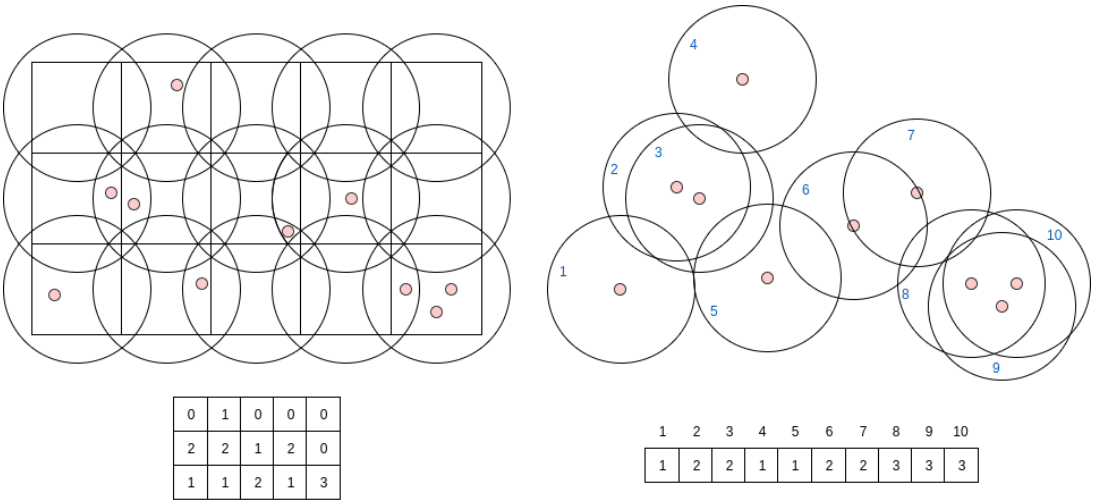


Figura 5: Método Grid vs. Método Kernel simplificados, únicamente se contabilizan los puntos que pertenecen a cada uno de los círculos generados

La Figura 5 muestra de forma visual los dos métodos de evaluación comentados y cómo, aún cuando los métodos de estimación de cada círculo han sido simplificados al número de puntos que contienen cada uno de los círculos, el total de ellos a evaluar y sus intersecciones difieren.

4.2. Búsqueda de estudios, investigaciones y datasets

Con el fin de cumplir uno de los objetivos marcados en el proyecto, analizar los parámetros que han sido utilizados en estudios reales e intentar reproducirlos, se realizó una búsqueda de estudios e investigaciones que utilizaran el método de KDE para extraer la información requerida:

1. Tamaño de celda
2. Ancho de banda o bandwidth
3. Dataset original para reproducir los resultados

En un inicio se implementó, para poder realizar pruebas base, un ejecutable en C++ que dados unos parámetros de entrada de la bounding box y el número de puntos deseados, genera un fichero que contiene tantos puntos como los deseados, generados de manera aleatoria con una distribución uniforme por toda la bounding box estipulada.

Con un total de 37 estudios sobre KDE analizados, de únicamente 15 fue posible extraer información parcial relativa a los parámetros requeridos. Más información en el apéndice de *Búsqueda de datasets*.

De todos ellos, únicamente en 11.1 se pudo encontrar el origen de los datos en los que se basó y pudieron ser obtenidos. Analizando los resultados de KDE para poder encontrar focos de accidentes en carreteras, el estudio utiliza los datos de UK Data Service [16], un servicio de datos de Reino Unido accesible para la investigación.

Con el objetivo de obtener los datos reales para poder analizar las implicaciones de estos en los resultados del proyecto, se realizó una instancia al servicio para poder tener acceso. El día 11 de setiembre de 2017 se obtuvieron las credenciales para acceder al servicio y así tener acceso a los datasets reales deseados.

Se descargaron los datos de accidentes de tráfico del año 2003 en todo UK y, a partir de ellos y después de analizar la *National Grid* [17], sistema de ubicación utilizado por todas las estadísticas geolocalizadas en Reino Unido, se generaron dos datasets de diferentes tamaños para poderlos analizar una vez implementadas las herramientas necesarias para el proyecto.

No se generaron los datasets originales del estudio puesto que la información que se definía en ellos para su generación no era suficiente, aún cuando definían las fechas de los datos analizados, no se definían los filtros aplicados en el dataset con el objetivo de acotarlo y, analizarlo por completo con el hardware del que se dispone no era viable para el proyecto.

Junto a la información real se dispuso, para realizar en el estudio distintos datasets, un conjunto de datasets variados obtenidos de la *University of Eastern Finland*[33]

Se han utilizado un total de **11** datasets:

- **2** datasets reales extraídos de la información de UK Data Service
- **8** datasets extraídos de *University of Eastern Finland*
- **1** dataset creado con la herramienta de generación de datasets de test.

De los 8 datasets extraídos de *University of Eastern Finland*:

- **4** datasets de datos bidimensionales sintéticos con $N = 5000$ puntos y $k = 15$ clústers gaussianos con diferente grado de superposición de clúster

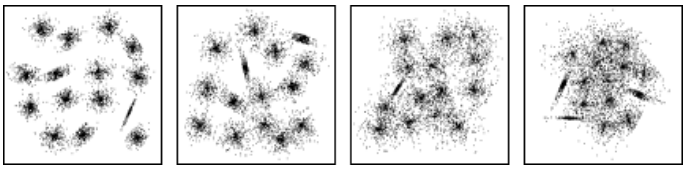


Figura 6: Visualización de los datasets $s1$, $s2$, $s3$ y $s4$ de [33]

- **3** datasets de datos bidimensionales sintéticos con números variables de puntos N y clústeres M. 150 puntos por cada clúster.

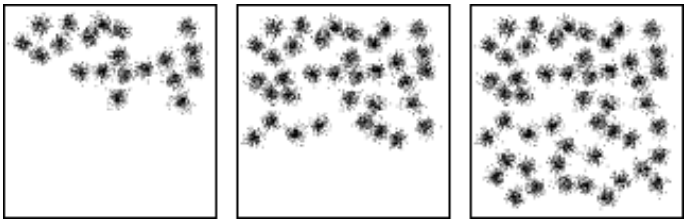


Figura 7: Visualización de los datasets $a1$, $a2$ y $a3$ de [33]

- **1** dataset de datos bidimensionales sintéticos con $N = 6500$ vectores y $k = 8$ clústeres gaussianos desequilibrados en número de puntos.

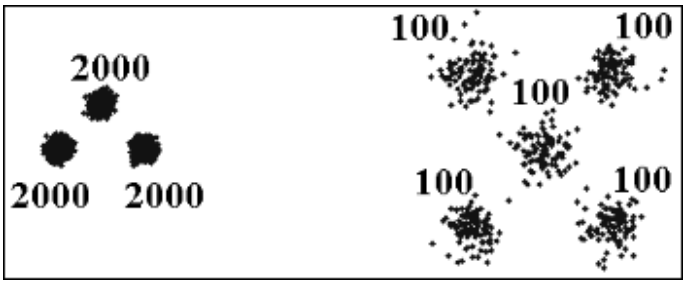


Figura 8: Visualización del dataset *unbalance* de [33]

Dado que únicamente se había podido obtener la fuente de los datos de un solo estudio, no se disponía de todos los parámetros necesarios para reproducirlos en el proyecto y los datasets finales a utilizar variaban en tamaño, extensión en área y complejidad, se tomó la decisión de realizar a la hora de implementar las herramientas de análisis, una búsqueda de información para determinar por cada uno de los datasets y de forma automática, el ancho de celda y de banda óptimos.

4.2.1. Determinación del tamaño de celda y bandwidth

Como los resultados de las investigaciones no han podido ser reproducidos a partir de los datos y añadida la característica de haber escogido datasets de diferentes tamaños y distribuciones, se ha decidido realizar una aproximación referente a esos valores.

Se ha hecho una búsqueda de métodos que puedan determinar los parámetros de forma automática [24, 25, 26] óptimos, pero tras haber analizado y valorado su posible repercusión a la hora de implementarlos en la planificación del proyecto, se ha determinado que la mejor opción es utilizar el método de Matlab con ksdensity, dada la rapidez a la hora de lograr una implementación:

"ksdensity estimates the density at 100 points for univariate data, or 900 points for bivariate data."

Dada la definición y aproximando la evaluación del número de puntos para datos bidimensionales, se ha decidido buscar la caja contenedora de cada uno de los datasets, escoger el menor de los dos lados (altura o anchura), y dividir dicho lado entre 30 unidades, pues se ha aproximado al número de puntos para datos bivariados que Matlab evalúa $\sqrt{900} = 30$

Para el bandwidth, se ha aprovechado del propio algoritmo ksdensity que retorna el valor óptimo del parámetro de forma automática.

4.3. Análisis de los datasets

Partiendo de la base de [1] y antes de definir los parámetros a analizar en el proyecto, los siguientes conceptos deben ser definidos puesto que determinan el coste computacional de los métodos propuestos:

- **Arreglo de círculos o circles arrangement:** Sea m el número de círculos generados por KDE, el arreglo de círculos se utilizará para extraer el número de intersecciones que se producen entre ellos.
- **Grafo de incidencias:** Grafo generado en base a las siguientes definiciones:
  - **Vértice:** Cualquier punto del dataset y cualquier centro de círculo son vértices.
  - **Arista:** Si existe la pertenencia de un punto p cualquiera a un círculo c cualquiera, existe una arista que une los vértices p y el centro del círculo c en el grafo.

Definidos los conceptos de complejidad, en el presente proyecto se analizan los siguientes parámetros:

- **Número de círculos generados:** Número total de círculos que se generan para evaluar KDE.
- **Número de intersecciones del arreglo de círculos:** Número total de intersecciones producidas en el arreglo de círculos.
- **Tamaño del grafo de incidencias:** Partiendo de la definición anterior del grafo de incidencias, el número de aristas que contiene.
- **Relación entre el bandwidth y el tamaño de celda, únicamente para el método Grid**

5. Diseño e implementación del software del proyecto

En este apartado se explicarán los diferentes elementos del software propuesto, su evolución a lo largo del proyecto y las posibilidades que se consideraron a la hora de su implementación.

5.1. Estructura de la solución

Para poder determinar los requisitos de la solución implementada, ha sido necesario evaluar cada una de las partes que componen la solución a los objetivos planteados en el proyecto, analizar qué tareas realiza cada una y determinar, de forma adecuada, las metas a alcanzar y la comunicación que a establecer entre ellas.

5.1.1. Requisitos de la solución

Objetivos a resolver:

1. Dados n puntos y m círculos en un plano, calcular el umbral r máximo de perturbación tal que el número de puntos interiores en cada círculo no cambie.
2. Dados n puntos, m círculos en el plano y un parámetro δ, calcular el umbral de perturbación máximo r de tal manera que el número de puntos en los círculos cambia en un porcentaje δ máximo.
3. Análisis de los resultados obtenidos con el método KDE variando los parámetros de entrada especificados, con el objetivo de evaluar si existe una relación entre los datos de entrada y de salida que permitan, a la hora de trabajar con datos geográficos, escoger el porcentaje máximo de desviación permitido en los resultados con datos enmascarados respecto a los resultados de los datos originales.

El objetivo 1 es una simplificación del objetivo 2, dado que si $\delta = 0\,\%$, el resultado obtenido por 2 será la solución al objetivo 1.

Elementos necesarios para la solución

1. Algoritmo que retorne los thresholds máximos para los dos métodos de la figura 5 y que puedan ser obtenidos bajo cualquier δ.
2. Visor de KDE
3. Herramienta de análisis de datasets
4. Herramienta de análisis de la variación del parámetro delta

Correspondiente a la búsqueda de la posible relación descrita en el problema 3, se realizarán análisis sobre los resultados y que serán especificados en la sección posterior referente a la herramienta de análisis de datasets y resultados.

5.2. Algoritmo para encontrar el threshold máximo

Se ha escrito un código en lenguaje C++, utilizando funciones de la librería de CGAL [11] con el objetivo de optimizar el código y reutilizar sus implementaciones.

En un inicio se planteó la posibilidad de poder implementar más de un algoritmo y así añadir a los análisis disponibles del proyecto, comparativas de tiempos de ejecución, dado que los diferentes algoritmos descritos no son sino optimizaciones alternativas al mismo problema.

Por motivos relacionados a la planificación y a los obstáculos encontrados a la hora de llevar a cabo los objetivos, únicamente se ha implementado el primer algoritmo descrito, el *Brute Force*.

5.2.1. Implementación del Brute Force

Como se definía en un apartado anterior, para encontrar el umbral máximo de perturbación r tal que el número de puntos en cada círculo sea modificado como mucho en factor δ en el algoritmo de Brute Force, se han de computar todos los puntos que recaen dentro de cada círculo.

Se define un umbral t_j para el cuál se calculan cuántos puntos pueden moverse dentro o fuera de C_j, buscando los puntos más cercanos a C_j en el interior y exterior.

Teórico:
Pudiéndose encontrar en tiempo $O(n)$, para todos los círculos toma una complejidad temporal de $O(nm)$.

Teorema 1 de [1]: Los problemas 1 y 2 pueden ser resueltos en tiempo $O(nm)$.

A la práctica:
En el aspecto práctico, como se verá en la sección relativa al código, la complejidad temporal de la implementación es de $O(mn * \log n)$.

Esto se produce por la búsqueda del parámetro δ, dado que se calculan todas las distancias $O(nm)$ y a la par se insertan de forma ordenada en una lista las distancias mínimas de cada punto al círculo más cercano para poder buscar, al finalizar, el índice correspondiente al threshold que cumpla el δ requerido, es decir:

$$index = \lfloor \delta * number\ of\ points \rfloor$$

Por tanto y dado que la lista de distancias mínimas está ordenado:

$$maxThreshold = sortedDistances[index]$$

5.2.2. Pruebas de validación del método Brute Force

Para validar el método implementado, se han definido dos funciones auxiliares dentro del código que realizan un test:

1. Sea tm el threshold máximo obtenido con el método descrito, se ha implementado una función de geomask que perturba cada uno de los puntos originales, aleatoriamente en distancia y dirección, y los almacena en una nueva lista.
2. Dado que los círculos están asociados internamente a un id único, su índice en la lista, se ha realizado una implementación de un método para crear una lista de tamaño igual al número de círculos que almacene en ella, de forma correlativa a los índices, el número de puntos que contiene cada uno
3. Se realiza el punto anterior para los puntos originales y los puntos perturbados
4. Se ha implementado otra función que recorre ambas listas en paralelo y comprueba, para cada par de círculos, si se supera la desviación máxima por encima del parámetro δ determinado.
5. En caso de superar el δ determinado, se dan los resultados como nulos y se devuelve un error al usuario.

La máxima desviación posible es determinada por:

$$maxDeviation = number\ of\ points * \delta$$

5.2.3. Mejoras realizadas a la implementación inicial

1. Función que realiza una búsqueda en el dataset para determinar el *Bounding Box* definido por los puntos y generar únicamente sus círculos internos. Únicamente sirve para el método Grid.
   Realizado
2. Ampliación del código del algoritmo para soportar iteraciones sobre el parámetro δ para ir incrementándose entre dos rangos de forma lineal.
   Realizado

5.2.4. Código relevante y comentarios

A continuación se destacarán las partes de código más relevantes del ejecutable *compute.x*.

Librerías CGAL utilizadas para el análisis de los círculos y la pertenencia de los puntos

```
// CGAL libraries
/* A model for Kernel that uses Cartesian
coordinates to represent the geometric objects. */
#include <CGAL/Cartesian.h>
#include <CGAL/bounding_box.h>
#include <CGAL/Circle_2.h>
#include <CGAL/Exact_rational.h>
#include <CGAL/squared_distance_2.h>    //for 2D functions
#include <CGAL/enum.h>

using namespace std;



typedef CGAL::Cartesian<CGAL::Exact_rational>   Kernel;
typedef Kernel::Circle_2                        Circle_2;
typedef Kernel::Point_2                         Point_2;
```

Se han aprovechado los objetos *Circle_2* y *Point_2* para utilizar funciones ya incluidas en la librería como las booleanas relativas a la pertenencia de un punto a un círculo, *circle.has_on_bounded_side*(*point*) o *circle.has_on_boundary*(*point*), que retornan booleano *True* o *False* en caso que se cumpla la condición.

Código principal

```
int main(int argc,  char** argv) {
    if(argc != 7) errorMsg();   // Check the number of parameters
    else {
        // Parse data
        ...

        // Get the bounding box of data
        getBoundaries(points, startX, endX, startY, endY);
        ...

        // Generate circles
        generateCirclesByGrid(startX, startY, rows, columns, cellSize,
          bandwidth*bandwidth, circlesGrid);
        generateCirclesByKernel(bandwidth*bandwidth, points, circlesKernel);
```

```
        // Get thresholds
        double r2Grid, r2Kernel;
        double h = (endDelta - initDelta) / (iterations - 1);
        double val = initDelta;
        for(int uint = 0; i < iterations; ++i) {
            r2Grid = bruteForceDelta(points, circlesGrid, bandwidth, val/100.0);
            r2Kernel = bruteForceDelta(points, circlesKernel, bandwidth, val/100.0);
            cout << r2Grid << endl;
            cout << r2Kernel << endl;
            val += h;
        }
```

Una vez se han leído correctamente los datos, se itera sobre los diferentes *deltas* y se muestra por la salida estándar el resultado de dichos cálculos.

Tests para los thresholds calculados

```
        // Randomize Points, Geomask
        vector<Point_2> newPointsByGridDelta;
        geomask(points, r2Grid, newPointsByGridDelta);
        vector<Point_2> newPointsByKernelDelta;
        geomask(points, r2Kernel, newPointsByKernelDelta);

        // Count points
        vector<int> countPointsInitialByGrid(circlesGrid.size(), 0);
        countPointCircles(points, circlesGrid, countPointsInitialByGrid);
        vector<int> countPointsGeomaskedByGridDelta(circlesGrid.size(), 0);
        countPointCircles(newPointsByGridDelta, circlesGrid,
                countPointsGeomaskedByGridDelta);
        vector<int> countPointsInitialByKernel(circlesKernel.size(), 0);
        countPointCircles(points, circlesKernel, countPointsInitialByKernel);
        vector<int> countPointsGeomaskedByKernelDelta(circlesKernel.size(), 0);
        countPointCircles(newPointsByKernelDelta, circlesKernel,
                countPointsGeomaskedByKernelDelta);

        bool test = true;
        double maxDesviation = points.size()*delta;
        for(int i = 0; i < circlesGrid.size() and test; ++i) {
            double desviationGrid = abs(countPointsInitialByGrid[i] - countPointsGeomaskedByG
            if(desviationGrid > maxDesviation) return 1;
        }
        for(int i = 0; i < circlesKernel.size() and test; ++i) {
            double desviationKernel = abs(countPointsInitialByKernel[i] - countPointsGeomasked
            if(desviationKernel > maxDesviation) return 1;
        }
    }
```

5.3. Visor de resultados

Partiendo del conocimiento previo del programador en el ámbito de los servidores y las plataformas web, se planteó en un primer momento el visor de resultados (visor de KDE), como una página dinámica donde el usuario pudiera introducir los parámetros necesarios y se visualizara de forma automática la distribución de probabilidades sobre los puntos originales.

Para ello, se realizó una búsqueda de librerías en Javascript que pudieran calcular y mostrar los resultados de KDE. Las librerías encontradas fueron:

- **D3.js:** Librería gráfica especializada en la visualización de datos. [14]
- **Plot.ly:** Librería que permite visualizar los resultados deseados. [15]

Tras escoger D3 como la librería que mejor se ajustaba a las necesidades del proyecto y por la claridad en la documentación, se creó un servidor NodeJS con ExpressJS a modo de API REST que renderizaba como única página un documento HTML que contenía todos los datos necesarios para mostrar el KDE.

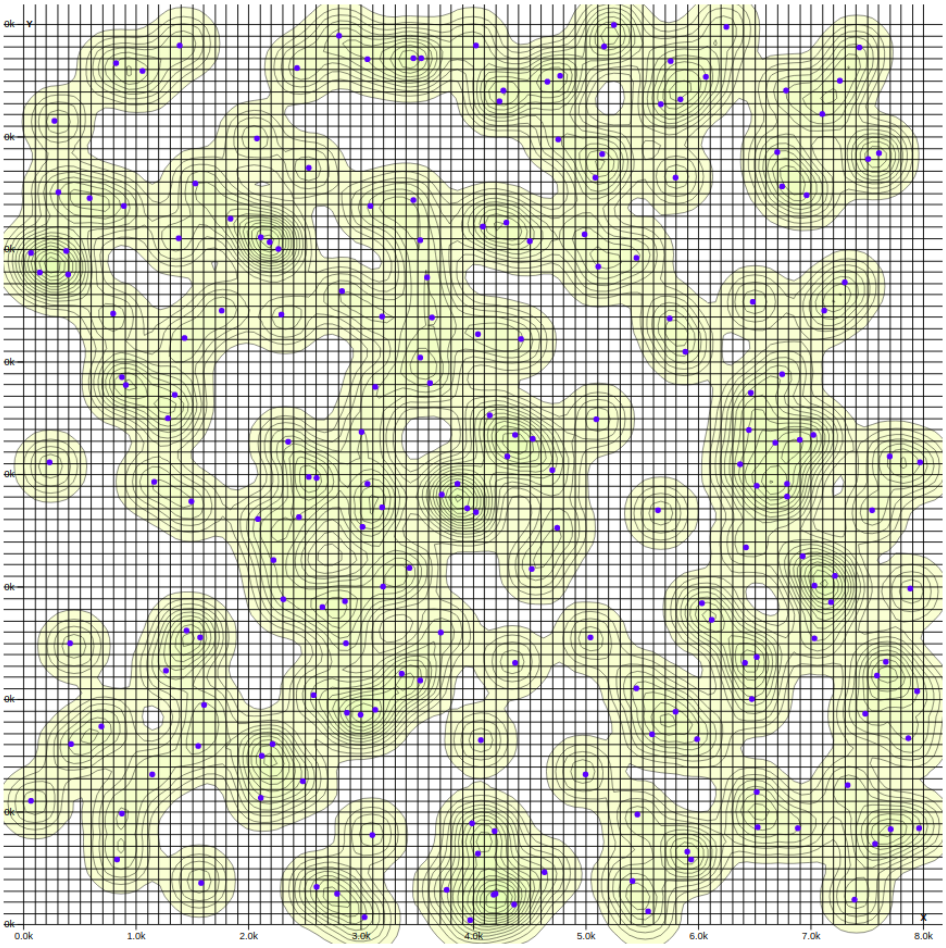


Figura 9: Visualización de los contornos de KDE con la librería D3.js

5.3.1. Problemas en la visualización de datos

La implementación de un visor en Javascript en un inicio fue ágil para realizar las primeras visualizaciones de datos. Más adelante, a la hora de modificar ciertos parámetros y añadir funcionalidades, las implementaciones se volvieron cada vez más costosas y lentas.

Finalmente, dadas todas las necesidades del visor y la futura agregación de las herramientas de análisis, se decidió por simplicidad realizar una reestructuración del visor para enfocarlo en el entorno de Matlab.

La decisión del entorno de Matlab fue inducida por el uso que se hace de ella en la asignatura de Visión por computador, VC.
Dado que el programador está cursando dicha asignatura, la utilizaba de forma habitual y su capacidad de cálculo, se decidió enfocar la parte de las herramientas de análisis y de visualización hacia ese entorno.

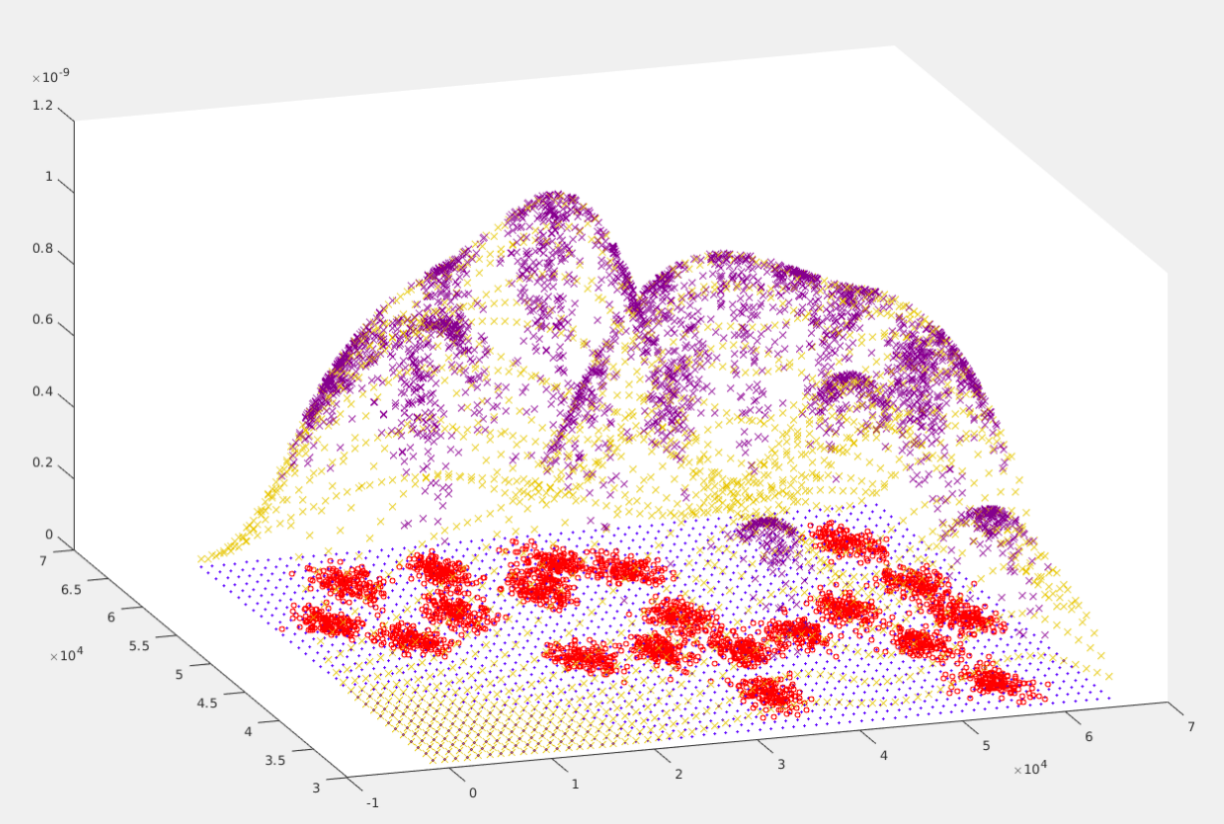


Figura 10: Visualización de KDE en 3D con Matlab, donde las coordenadas definen el plano acotado por los ejes X e Y y el eje Z viene definido por el valor de probabilidad evaluado en cada punto determinado. Puntos originales en rojo, KDE con el método Grid en amarillo, KDE con el método Kernel en violeta.

5.4. Herramienta de análisis de datasets

La herramienta de análisis de datasets se empezó a implementar tras el cambio de tecnología de Javascript a Matlab. Por tanto, no se produjeron cambios en la implementación de la herramienta.

Para los métodos Grid y Kernel, los parámetros que se evalúan para cada dataset tal como se define en la Sección 4.3 son los siguientes:

- Número total de círculos generados.
- Número de intersecciones en el arreglo de círculos generados.
- Tamaño del grafo de incidencias generado.
- Relación entre el bandwidth y el tamaño de celda, únicamente para el método Grid.

Se ha implementado un código en C++ y CGAL llamado *analysis.cpp* que, dado un nombre de dataset y los parámetros a evaluar, devuelve toda la información relativa al análisis de los datasets mencionada.

A continuación se destacarán las partes de código más relevantes del ejecutable *analysis.x*.

```
// CGAL imports
/* A model for Kernel that uses Cartesian
coordinates to represent the geometric objects. */
#include <CGAL/Cartesian.h>
#include <CGAL/bounding_box.h>
#include <CGAL/Circle_2.h>
#include <CGAL/Exact_rational.h>
#include <CGAL/squared_distance_2.h> //for 2D functions
#include <CGAL/enum.h>
#include <CGAL/Arr_circle_segment_traits_2.h> // CGAL circle arrangements
#include <CGAL/Arrangement_2.h>

using namespace std;



typedef CGAL::Cartesian<CGAL::Exact_rational>          Kernel;
typedef Kernel::Circle_2                               Circle_2;
typedef Kernel::Point_2                                Point_2;

// Arrangements
typedef CGAL::Arr_circle_segment_traits_2<Kernel>      Traits_2;
typedef Traits_2::Curve_2                              Curve_2;
typedef Traits_2::CoordNT                              CoordNT;
typedef CGAL::Arrangement_2<Traits_2>                  Arrangement_2;
```

Se ha aprovechado la librería CGAL para importar las librerías relacionadas con el arreglos de objetos en 2 dimensiones.[27, 28].
Cabe remarcar que para poder introducir de manera correcta los círculos al arrangement, hace falta realizar una conversión de los círculos a tipo *Curve_2* de CGAL y, tal como se define la propia documentación:

"Suppose that we construct an arrangement of circles. A circle is obviously not x-monotone, so we cannot insert it in the same way we inserted x-monotone curves. However, it is possible to subdivide each circle into two x-monotone circular arcs (its upper half and its lower half) and to insert each x-monotone arc separately."[27]

Por esta segunda causa, y siguiendo el procedimiento descrito en [29] dado que un círculo se producirán exactamente 2 intersecciones, el total de intersecciones será igual a:

$$Number\_of\_intersections = arrangement.number\_of\_vertices() - circles.size() * 2$$

Para calcular el tamaño del grafo de incidencias, se itera sobre todos los círculos y sobre todos los puntos y se realiza un sumatorio de todas las pertenencias a círculos por cada uno de los puntos.

5.5. Herramienta de análisis de la variación del parámetro delta

De igual forma que la herramienta de análisis de datasets, la herramienta de análisis de resultados empezó a desarrollarse a fecha posterior del cambio de tecnología en el visor. Por este motivo, no se produjeron cambios en la implementación de la herramienta.

Para los métodos Grid y Kernel, existen dos tipos de pruebas a realizar en la herramienta:

- Dado un δ especificado por el usuario:
    1. Ejecutar el algoritmo implementado en *compute.x* y obteniendo los thresholds máximos *tmg* y *tmk* respectivamente.
    2. Se perturban en radio máximo (para ver los resultados en el peor de los casos) los puntos en función de dichos thresholds obtenidos para cada método de evaluación y se extraen densidades por ksdensity [26]
    3. Se repite el paso anterior 30 veces y se calcula la media para cada uno de los puntos evaluados.
    4. Una vez obtenida la densidad media de perturbación, se presentan los resultados con un gráfico Quantile-Quantile de Matlab respecto a las densidades de los puntos originales para visualizar las desviaciones.

- Dado un rango de δ's y un número especificado de iteraciones por el usuario:

    1. Ejecutar el algoritmo implementado en *compute.x* y obteniendo los thresholds máximos *tmg* y *tmk* respectivamente del método Grid y Kernel tantas veces como iteraciones se hayan marcado en la ejecución, siendo δ incrementado.

    2. Por cada par de thresholds obtenidos en la ejecución, se perturban en radio máximo (para ver los resultados en el peor de los casos) los puntos en función de dichos valores obtenidos para cada método de evaluación y se extraen densidades por ksdensity [26]

    3. Se repite el paso anterior 30 veces y se calcula la media para cada uno de los puntos evaluados.

    4. Una vez obtenidas la densidades medias de perturbación por cada uno de los δ's, se ejecuta el test de Kolmogorov-Smirnov [30] que calcula la diferencia máxima entre dos puntos de una distribución y, bajo la formulación del mismo, la similaridad entre las densidades estimadas. Aún cuando el test para datos de más de una dimension puede implicar errores como se cita en [31], se ha utilizado su implementación para evaluar únicamente en los puntos en los que ksdensity ha evaluado y, por tanto, se tratan como listas de probabilidades discretas ordenadas bajo el mismo método.

5.6. Estructuras planteadas y estructura final del software

Tal como se ha comentado en las diferentes partes a implementar del proyecto, surgieron errores y problemas que provocaron la modificación de la estructura en la que las partes se comunicaban.

En la fase inicial, con el visor en Javascript, se estructuró la solución como se muestra a continuación:

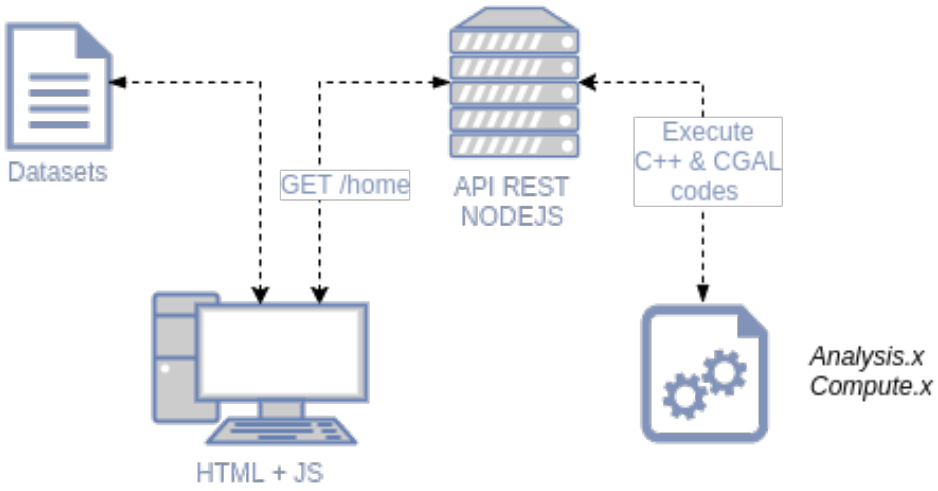


Figura 11: Estructura inicial de la solución

El servidor estaba como intermediario entre los diferentes módulos y se utilizaba a modo de API server. Por dificultades referentes a la claridad del código, las propias librerías utilizadas para la web y la complejidad a la hora de poder implementar las herramientas de análisis, se realizo un cambio por completo en la visualización.

Para solucionar los problemas encontrados y no entorpecer más tarde la programación, se decidió reestructurar toda la solución como se muestra a continuación:

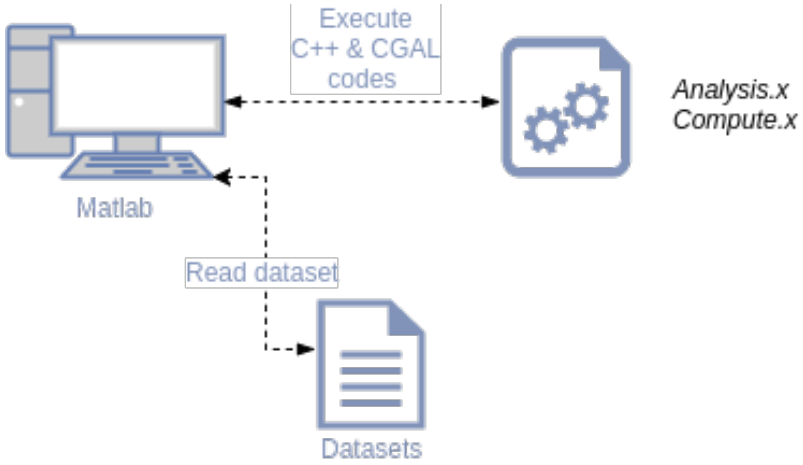


Figura 12: Solución reestructurada

Como optimización a la hora de transmitir los datos, cuello de botella en las ejecuciones dependiendo del dataset, se decidió añadir al código C++ una función que dado el nombre de un dataset cualquiera, lo lea directamente del fichero donde está almacenado.

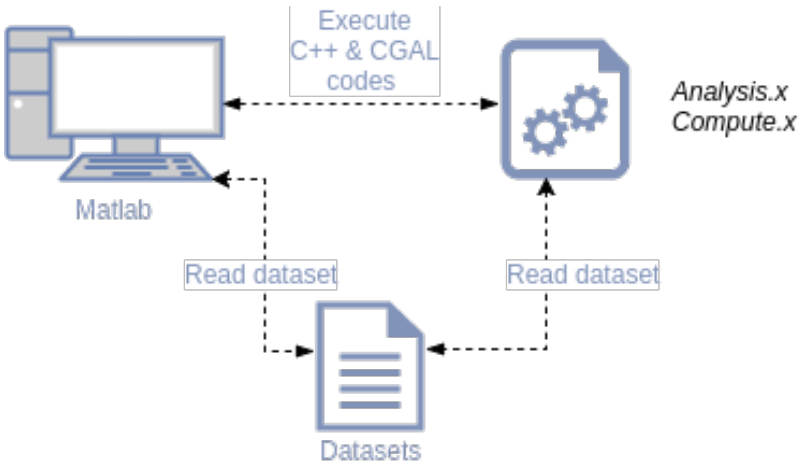


Figura 13: Estructura final de la solución

Las partes de la solución definida son:

1. Archivos de datasets en formato de texto plano que definen puntos en el espacio 2D.
2. GUI en matlab a modo de herramienta para el usuario para la visualización, análisis de datasets y análisis de los resultados obtenidos.
3. Código C++ que realiza todos los cálculos relacionados con el análisis de datasets.
4. Código C++ que realiza todos los cálculos relacionados el threshold máximo bajo los dos métodos de evaluación para un dataset y un intervalo de δ's determinado.

6. Resultados

Tal como se ha explicado en las secciones anteriores 2.1 y 4.3, se han ejecutado los tests para los 11 datasets disponibles en el trabajo (Más información sobre los datasets en la Sección 4.2).

A continuación se presentan los resultados obtenidos.

6.1. Análisis de datasets

Dataset	# of points	CellSize (m)	Area (km²)	Area(km²) / # of points	Bandwidth, bw (m)	Bw/cellSize	Type	Total Circles	Intersections of the circles arrangement	Size of the incidences graph
Dataset 1 real	294	586,667	0,050512	0,00017181	175,358	2,989	Grid	1600	2013	4844
						-	Kernel	294	5494	3454
Test data	400	264,567	63,41663	0,158541575	1074,62	4,0618	Grid	1023	171760	19284
						-	Kernel	400	27816	8378
Dataset 2 real	2126	19,8	0,427086	0,000200887	277,012	1,399	Grid	1216	22800	11370
						-	Kernel	2126	190656	89204
A1	3000	1097,13	2157,01899	0,71900633	2316,57	2,1115	Grid	2013	103468	20849
						-	Kernel	3000	495040	221210
S1	5000	30654,5	866402,8477	173,2805695	72291,4	2,3583	Grid	1024	61168	87210
						-	Kernel	5000	1945270	1294320
S2	5000	30933,4	889882,4309	177,9764862	64232,4	2,0765	Grid	1023	51028	67617
						-	Kernel	5000	1896352	985072
S3	5000	29244	798024,2768	159,6048554	59132,4	2,022	Grid	1056	45596	64095
						-	Kernel	5000	2191222	808036
S4	5000	28111,7	816040,6403	163,2081281	48130,2	1,7121	Grid	1147	37816	45335
						-	Kernel	5000	1798592	685304
A2	5250	1458,4	2867,28732	0,546149966	3450,27	2,3658	Grid	1472	89728	65499
						-	Kernel	5250	1520346	661172
Unbalanced	6500	5647	73867,16466	11,36417918	1679,91	0,29749	Grid	2528	0	1645
						-	Kernel	6500	3025944	846820
A3	7500	2105,53	4139,58381	0,551944508	5200,92	2,4701	Grid	1089	65316	143487
						-	Kernel	7500	4554236	1424700

Figura 14: Resultados de los parámetros específicos de los diferentes datasets

Observaciones: Como se puede ver en la Figura 11 en el dataset *Unbalanced*, el método definido para encontrar el ancho de banda y el bandwidth no funciona correctamente, pues el bandwidth es menor al cellsize y, por tanto, en el método grid no existirán intersecciones en el arreglo de círculos. Dicho error produce que pueden existir puntos en el plano que no estén considerados por ninguno de los círculos y por tanto por el KDE.

6.1.1. Relación entre el número de puntos y el número de intersecciones del arreglo de círculos

En las siguientes gráficas se muestra la evolución del número de puntos (eje X) respecto al número de intersecciones en el arreglo de círculos (eje Y).

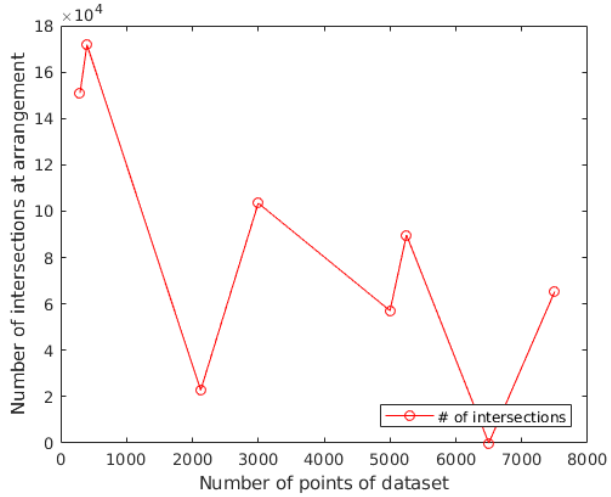


Figura 15: Número de puntos/Número de intersecciones del arreglo de círculos para el método Grid. Los datasets han sido ordenados por el número de puntos que contienen, tal y como se han analizado en la tabla anterior

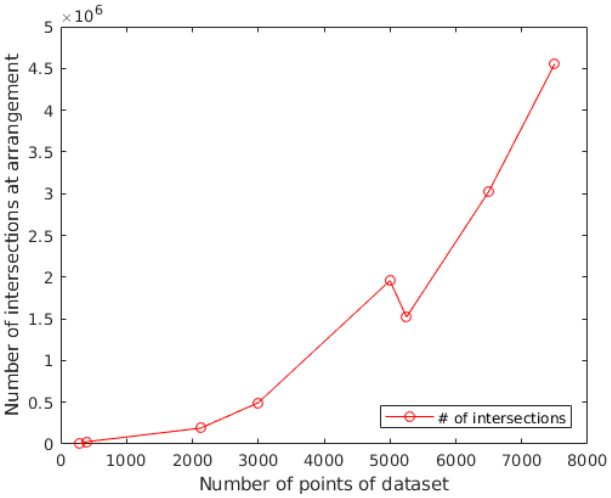


Figura 16: Número de puntos/Número de intersecciones del arreglo de círculos para el método Kernel. De igual forma que la gráfica anterior, los datasets han sido ordenados por el número de puntos que contienen, tal y como se han analizado en la tabla anterior

Observando las gráficas generadas, se puede detectar que aunque el método Grid no parece seguir ningún tipo de relación entre los parámetros de número de puntos e intersecciones en el arreglo.
Tal comportamiento es esperable si se toma en cuenta que la solución a la búsqueda de dicho parámetro en este proyecto ha sido utilizar un valor fijado e independiente de la densidad del dataset, el área que abarque la bounding box o el bandwidth utilizado.

Por otra parte sí parece que bajo el método Kernel puede existir una relación lineal entre estos dos parámetros.

6.1.2. Relación entre el número de puntos y el tamaño del grafo de incidencias

En las siguientes gráficas se muestra la evolución del número de puntos (eje X) respecto al tamaño del grafo de incidencias (eje Y).

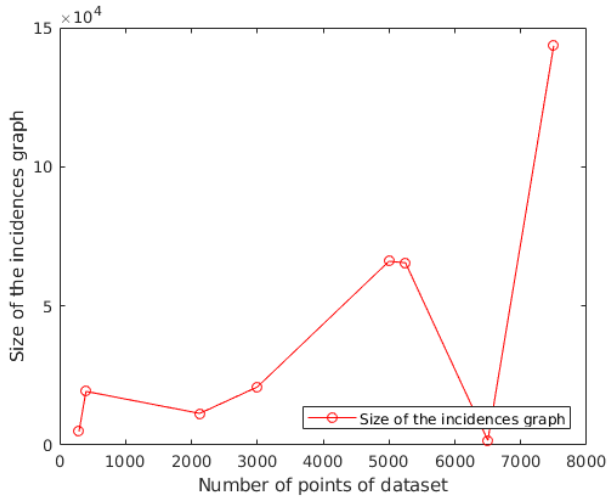


Figura 17: Número de puntos/Tamaño del grafo de incidencias para el método Grid.
Número de puntos de cada uno de los datasets, siendo estos ordenados por dicho parámetro

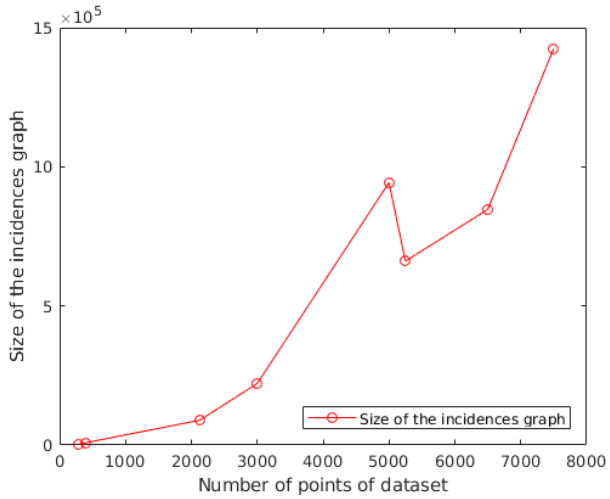


Figura 18: Número de puntos/Tamaño del grafo de incidencias para el método Kernel.
Número de puntos de cada uno de los datasets, siendo estos ordenados por dicho parámetro

De nuevo, por los mismos motivos definidos anteriormente, se puede detectar que aunque el método Grid no parece seguir ningún tipo de relación entre los parámetros de número de puntos y el tamaño del grafo de incidencias, y sí parece que bajo el método Kernel puede existir una relación lineal entre estos dos parámetros.

6.2. Análisis de thresholds

Se ha realizado una búsqueda de los thresholds máximos para cada uno de los datasets variando el parámetro δ de forma iterativa, tomando dicho parámetro los siguientes valores porcentuales:

$$\delta = [0, 5, 10, 15, 20, 30, 50, 70, 90]$$

Dataset	Maximum threshold (m)								
	δ = 0%	δ = 5%	δ = 10%	δ = 15%	δ = 20%	δ = 30%	δ = 50%	δ = 70%	δ = 90%
A1	0.032059	1138.49	2990.84	4433.47	5930.53	8242.38	13401.2	21253.7	25784.3
	0.001943	1136.45	2573.23	4162.8	5799.57	8222.4	13637.3	18065.7	24870.1
A2	0.022319	1053.82	3517.94	5939.99	7538.35	10784.2	16024.4	20949.7	26037.9
	0.00028985	1160.6	3553.94	5829.97	7531.01	10938.2	15987.1	20946.2	25994.1
A3	0.0026442	1384.64	4061.85	6327.18	8714.31	12642.9	18683.9	23410.9	28399.3
	0.00048077	1303.51	3979.54	6351.03	8631.75	12591.4	18753.4	23570.9	28454.8
S1	0.40094	7913.85	31594.6	75421	91266.2	159340	237700	311680	362010
	0.041422	15546.9	35892.6	85640.7	103964	157156	235590	313184	371543
S2	0.40786	13469.8	38894.9	65817.6	103023	156949	233207	307972	365125
	0.073458	15086.1	39890.6	73871.1	103213	156086	232451	307062	363240
S3	0.31096	7599.74	28227	53522.8	72736.5	113884	190080	258984	340206
	0.052194	7097.28	27757.3	52870	70534.5	113357	190073	258676	339509
S4	0.0095922	9825.44	35878.8	65236.5	84031.5	123870	177478	223453	272550
	0.022772	8213.04	35267.2	64621.9	83227	120887	176338	222976	270624
Unbalanced	0.84355	674.439	1399.38	2361.75	3436.5	7770.77	22550.1	30234.4	35482.4
	0.0085148	511.55	1125.22	1859.94	2761.22	6570.1	29615.7	30575.5	40177.3
Test data	0.012301	104.893	306.683	540.555	809.844	1277.42	2024.66	2653.67	3317.97
	0.033432	132.261	303.781	600.578	835.639	1319.56	2065.16	2676.3	3343.52
Dataset 1 real	0.00009924	1.554	5.0296	9.3042	15.9267	29.6435	56.1338	74.6476	108.082
	0.042944	1.6996	5.551	11.7048	17.3927	30.5683	54.4711	76.3844	108.995
Dataset 2 real	0.0004245	8.3554	20.1747	34.9855	47.559	75.362	123.452	178.808	258.787
	0.029649	6.6062	18.8284	32.7389	46.5484	74.4751	123.02	178.097	257.486

Figura 19: Resultados de los thresholds máximos para cada dataset en función de δ

Tal como se puede ver en la tabla, la relación entre threshold máximo y δ parece ser lineal para los dos tipos de metodologías de evaluación, Grid y Kernel.

Con el objetivo de comprobar de una forma visual dichas asunciones, se han generado unas gráficas que relacionan el threshold máximo (eje X) en función del parámetro δ (eje Y) para cada dataset.

6.2.1. Relación entre threshold máximo y el parámetro δ asociado

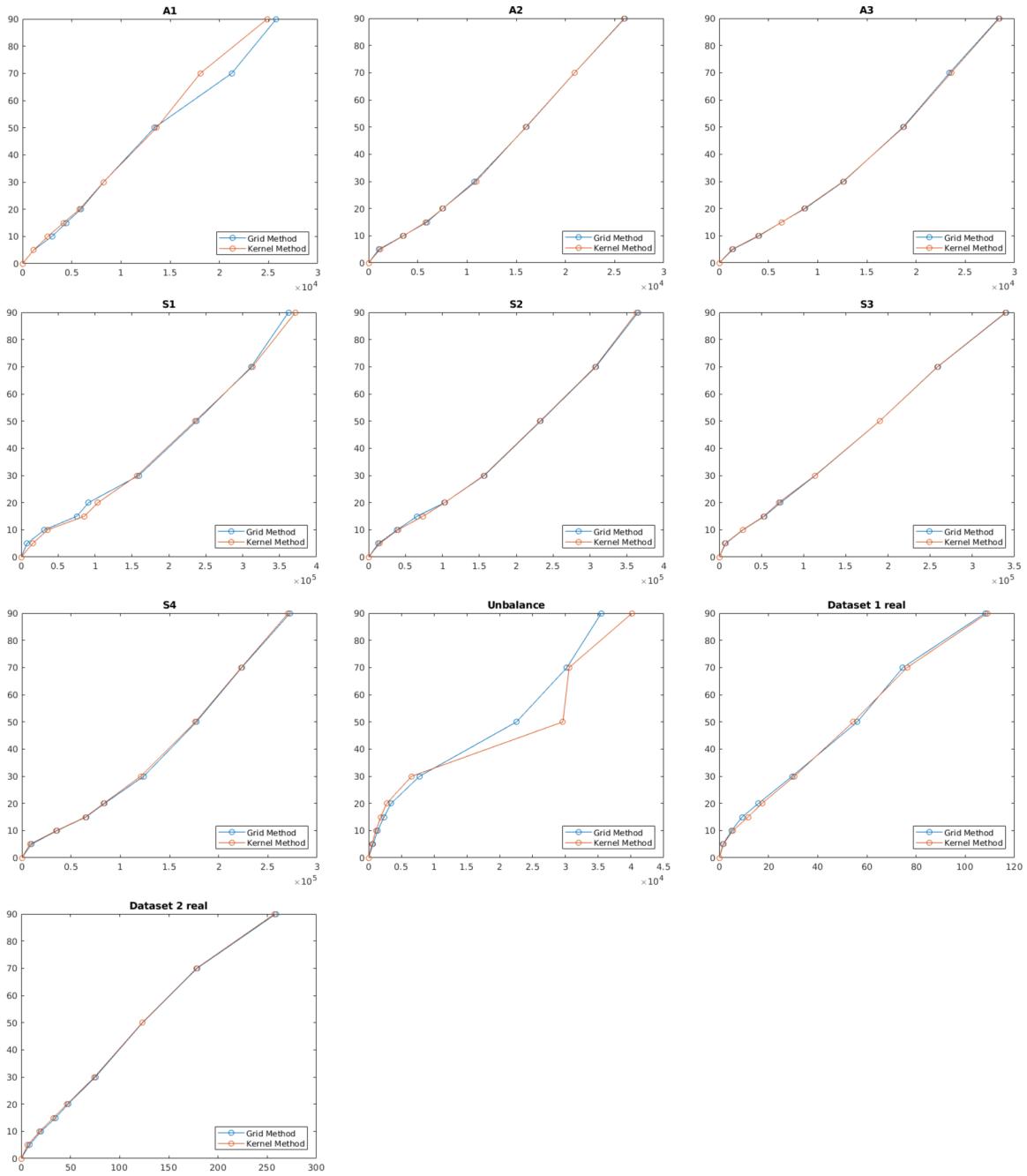


Figura 20: Gráficas de los thresholds máximos para cada dataset en función de δ

Tal como se había planteado, la relación entre el parámetro de desviación δ y el threshold máximo, es lineal. La única relación que está desviada es la que pertenece al dataset *Unbalanced*.

El dataset *Unbalanced* tiene la característica de tener pocos clústers y muy separados entre ellos. Como consecuencia, encontramos cómo en el método Kernel el threshold se ve incrementado de

manera significativa al permitir un desvío δ suficiente para poder desviarse los puntos entre clústers. En el caso del dataset comentado, a partir del 30 % el threshold aumenta considerablemente.

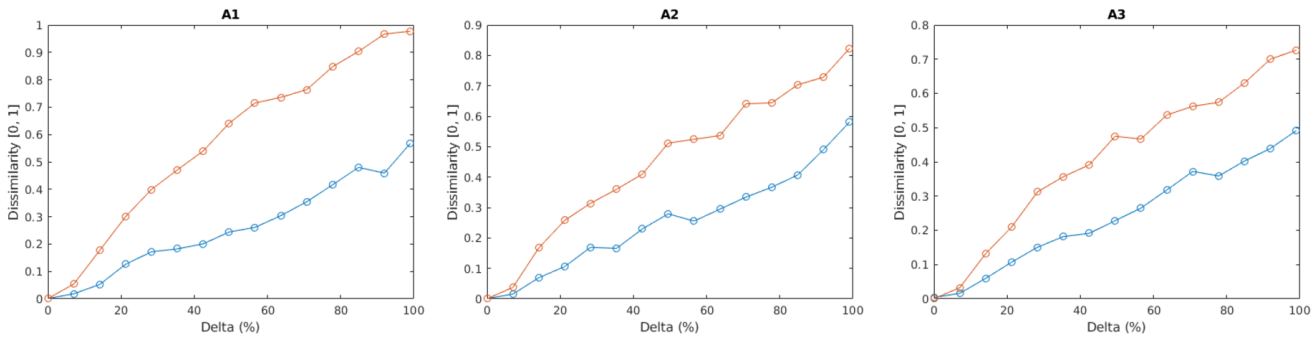
6.3. Análisis de la variación de delta

Tal como fue definido en la sección 5.5, a continuación se muestran los resultados obtenidos de modificar iterativamente en un rango determinado el test de Kolmogorov-Smirnov [30], que comprueba el grado de similaridad entre dos distribuciones.

A1		A2		A3		S1		S2		S3	
Grid	Kernel	Grid	Kernel	Grid	Kernel	Grid	Kernel	Grid	Kernel	Grid	Kernel
0.0010	0.0007	0.0007	0.0002	0.0038	0.0003	0.0010	0.0004	0.0010	0.0004	0.0009	0.0004
0.0184	0.0540	0.0156	0.0377	0.0161	0.0325	0.0078	0.1124	0.0127	0.0736	0.0066	0.0178
0.0523	0.1763	0.0693	0.1672	0.0597	0.1321	0.1006	0.6234	0.0488	0.2734	0.0350	0.1156
0.1260	0.3000	0.1060	0.2577	0.1070	0.2095	0.1367	0.7350	0.0791	0.4084	0.0597	0.2144
0.1716	0.3993	0.1692	0.3141	0.1506	0.3124	0.1904	0.8014	0.1270	0.5222	0.0994	0.2578
0.1824	0.4707	0.1658	0.3598	0.1818	0.3568	0.2168	0.8302	0.1768	0.5974	0.1203	0.3306
0.2008	0.5397	0.2296	0.4097	0.1913	0.3913	0.2158	0.8330	0.1719	0.6306	0.1496	0.3774
0.2439	0.6410	0.2792	0.5116	0.2282	0.4748	0.2412	0.8630	0.2002	0.6394	0.1884	0.4134
0.2608	0.7150	0.2554	0.5244	0.2652	0.4669	0.2520	0.8194	0.1729	0.7014	0.2150	0.4544
0.3038	0.7357	0.2948	0.5364	0.3182	0.5373	0.2744	0.8720	0.2314	0.6992	0.2206	0.4696
0.3545	0.7640	0.3342	0.6408	0.3722	0.5624	0.2930	0.8732	0.2734	0.7662	0.2377	0.5224
0.4165	0.8477	0.3668	0.6440	0.3589	0.5741	0.2725	0.8822	0.2510	0.7304	0.2462	0.5390
0.4800	0.9033	0.4069	0.7034	0.4025	0.6309	0.3320	0.9068	0.2520	0.7090	0.2566	0.7072
0.4590	0.9670	0.4905	0.7284	0.4394	0.7003	0.3594	0.9154	0.2930	0.7654	0.2623	0.7324
0.5676	0.9770	0.5808	0.8229	0.4915	0.7265	0.3877	0.9612	0.3564	0.8866	0.3371	0.8158

S4		Unbalance		Test data		Dataset 1 Real		Dataset 2 Real	
Grid	Kernel	Grid	Kernel	Grid	Kernel	Grid	Kernel	Grid	Kernel
0.0009	0.0002	0.0008	0.0005	0.0010	0.0050	0.0006	0.0170	0.0008	0.0042
0.0104	0.0358	0.0059	0.0268	0.0244	0.0375	0.0094	0.0510	0.0107	0.0165
0.0443	0.2290	0.0119	0.0991	0.0420	0.0425	0.0281	0.0476	0.0238	0.0673
0.0799	0.3172	0.0237	0.2946	0.0518	0.0650	0.0281	0.0850	0.0493	0.1364
0.1250	0.3498	0.0400	0.5088	0.0732	0.1575	0.0419	0.1463	0.0732	0.2328
0.1580	0.3638	0.1748	0.8983	0.1289	0.1325	0.0844	0.2449	0.1118	0.2479
0.1858	0.3554	0.1784	0.8992	0.1641	0.1925	0.0756	0.3231	0.1283	0.2893
0.2031	0.3950	0.1883	0.8998	0.2305	0.2050	0.2000	0.3878	0.1414	0.3311
0.2465	0.4578	0.2045	0.8989	0.2178	0.2150	0.1994	0.5204	0.1785	0.4111
0.2622	0.4644	0.2544	0.9014	0.3760	0.2575	0.1856	0.4592	0.1933	0.4426
0.2778	0.5534	0.2654	0.9032	0.2168	0.2950	0.1737	0.6088	0.2196	0.4948
0.2917	0.5290	0.2820	0.9034	0.3086	0.3600	0.3419	0.7789	0.2533	0.5320
0.3108	0.6028	0.2856	0.9054	0.2715	0.3875	0.2175	0.7347	0.2738	0.6143
0.3290	0.6094	0.3279	0.9062	0.3389	0.5300	0.5012	0.7007	0.3109	0.6966
0.3828	0.8014	0.7449	0.9898	0.4707	0.5000	0.4938	0.8401	0.3470	0.8434

Figura 21: Tablas de los resultados del test de Kolmogorov-Smirnov



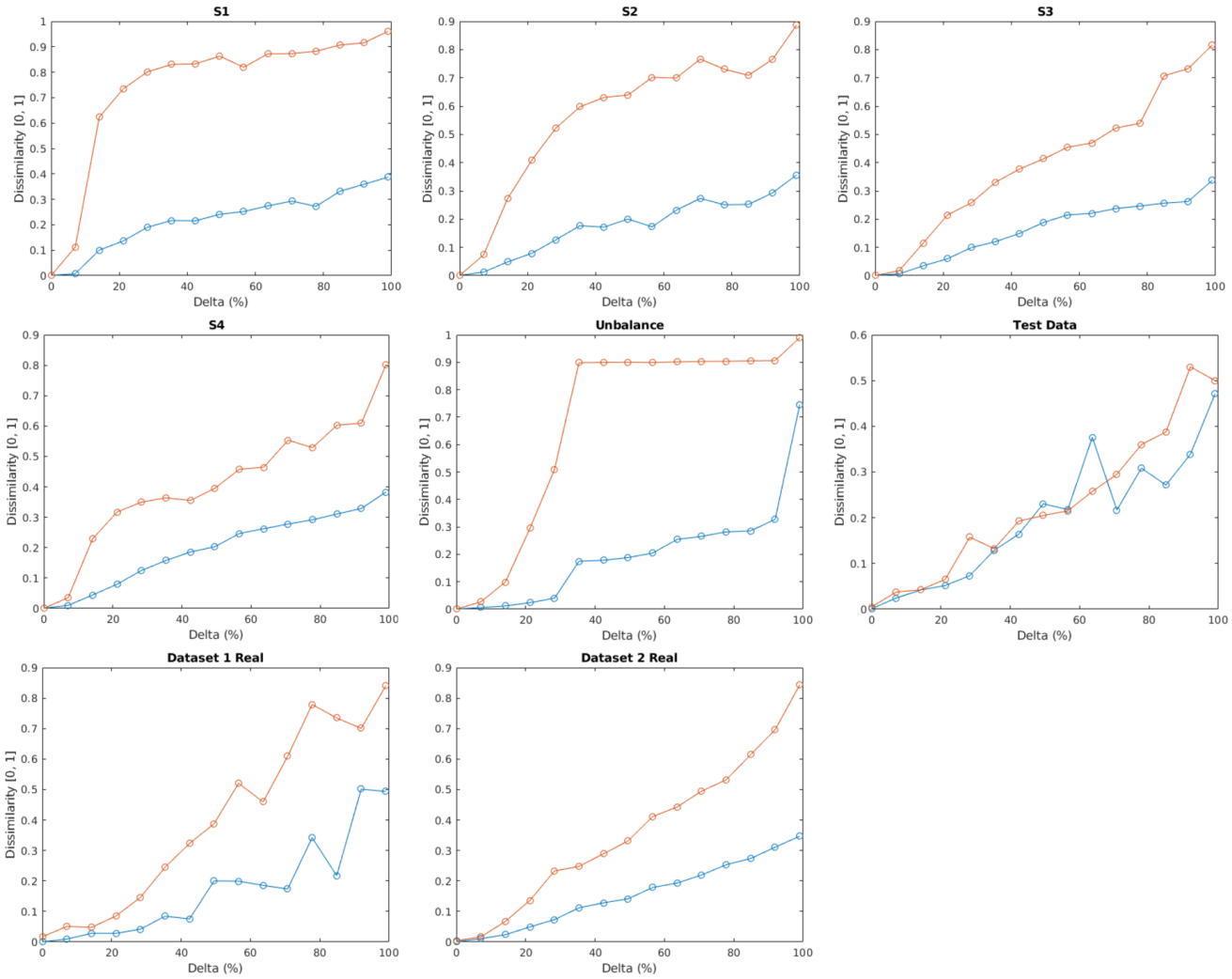


Figura 22: Gráficas de los resultados del test de Kolmogorov-Smirnov.
Las marcas azules hacen referencia al método de evaluación de Grid y las marcas naranjas al método Kernel.

Como se puede apreciar en los resultados, el método Grid sigue una relación aproximadamente lineal entre el porcentaje de variación de los puntos y la comparativa del test de Kolmogorov-Smirnov. Más sensible a las variaciones en δ, el método Kernel alcanza altos valores de no similaridad en bajos valores de variación.

7. Gestión del proyecto

En esta sección de detallan los posibles obstáculos que se tuvieron en cuenta a la hora de desarrollar el proyecto y la metodología a utilizar para realizarlo.

7.1. Posibles obstáculos, riesgos y alternativas

Falta de información en estudios e investigaciones en aspectos como el bandwidth o el tamaño de celda de las matrices para los algoritmos KDE
Alta probabilidad

Para el tamaño de celda o el bandwidth, se investigará de funciones que sean utilizadas con el fin de aproximar tales parámetros y obtener resultados que puedan asemejar estudios.

Imposibilidad de acceder a los datos originales de los estudios o investigaciones
Alta probabilidad

Se creará una herramienta que genere puntos aleatorios de forma uniforme sobre una región y sirva para generar datasets que puedan ser utilizados en el análisis. Por otra lado, se buscarán fuentes de datasets que no sean reales y ofrezcan datos con diferentes distribuciones de puntos para así estudiar de una forma más genérica los resultados obtenidos.

Heterogeneidad en el formato de los datos obtenidos
Alta probabilidad

Trabajando con un algoritmo que ha sido utilizado en diferentes campos científicos y tecnológicos, la heterogeneidad de los datos que proporcionan las fuentes es muy probable.

Por este motivo, si el caso se produce, se creará una herramienta que estandarice para la solución propuesta, los datos que se le proporcionen y los almacene de forma adecuada para el sistema.

Errores en el código
Probabilidad media

En un proyecto con una parte computacional compleja es necesario minimizar el número de errores, dado que se pueden llegar a generar resultados que difieran a los esperados. Con el objetivo de minimizarlos, se creará un sistema de pruebas para realizar tests sobre los resultados y comprobar que son los esperados.

Calendario
Probabilidad media

Debido al tiempo limitado de tres meses del proyecto, la calendarización del proyecto deberá ser estricta y cumplirse a objetivos definidos cada dos semanas para que el director del proyecto pueda validar el trabajo realizado en ese período y así guiar al alumno.

7.2. Metodología

7.2.1. Métodos de trabajo

Dado que el tiempo es limitado y el calendario vendrá definido por iteraciones de dos semanas, será necesario que se apliquen métodos de trabajo ágiles.

Siguiendo una metodología de trabajo de ciclos cortos (objetivos a alcanzar en períodos de dos semanas), se garantizará una visión real del estado del proyecto y se podrá verificar si se encuentra o no dentro del calendario estipulado del proyecto.

Por la dificultad del proyecto y el carácter que tienen en ese aspecto los posibles errores de programación, se utilizará una metodología de desarrollo basada en pruebas para minimizar la probabilidad de fallos.

Como conclusión y una vez analizados los aspectos de requerimientos que este proyecto necesita, se concluye que se utilizará la metodología Scrum [12].

7.2.2. Herramientas de seguimiento

La comunicación del seguimiento del proyecto se realizará preferentemente mediante las reuniones de seguimiento que están planificadas cada dos semanas en el transcurso del proyecto y, en caso de no poder realizarla, el uso del correo electrónico como canal comunicativo.

7.2.3. Métodos de validación

Para validar cada una de las secciones del proyecto, se mostrará el trabajo realizado al director del proyecto, quien aceptará o pedirá realizar modificaciones sobre el código o el análisis.

Durante el transcurso del proyecto, se ofrecerá feedback y se propondrán modificaciones en caso de ser necesario.

7.2.4. Herramientas para el desarrollo

Programación

Para desarrollar el código se utilizará Sublime Text, un editor de texto abierto y gratuito por un tiempo limitado que dispone de gran cantidad de packages y plugins que permiten visualizar y anticipar posibles errores de sintaxis en los códigos. [20] [21]

Control de versiones

Se realizará un control de versiones de código. Utilizando la cuenta para estudiantes que GitHub ofrece, se creará un repositorio Git privado en el que se irá actualizando el código implementado.

El control de versiones estará basado en un modelo de ramas [22] para posibilitar el trabajo paralelo de algunos de los aspectos del trabajo que son independientes entre sí.

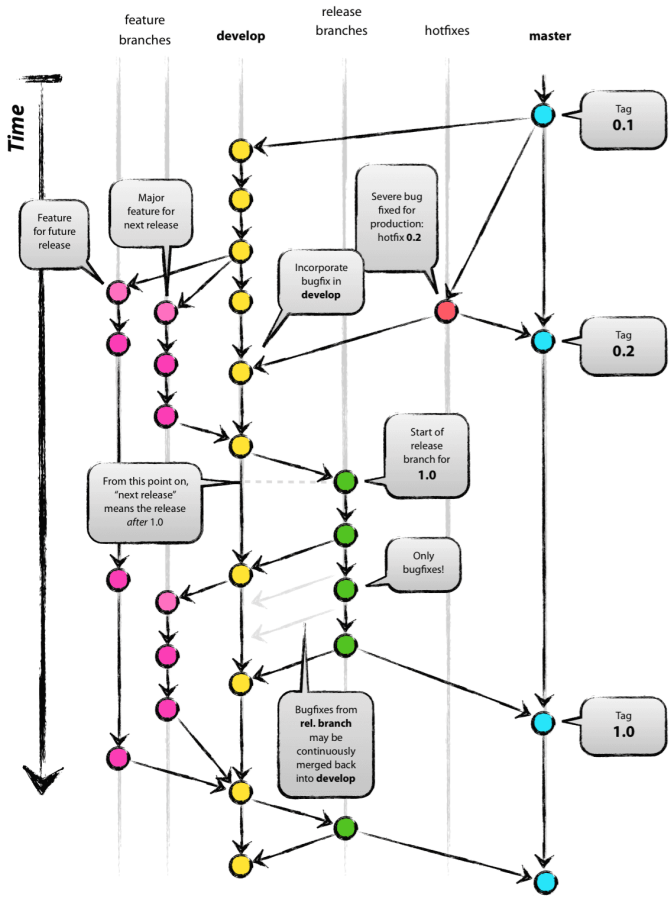


Figura 23: Ejemplo de control de versiones basado en ramas [22]

8. Planificación del proyecto

A continuación se añade la documentación relativa a la planificación que se realizó.

8.1. Delimitación temporal

El proyecto está acotado a una duración temporal de 470h (+5 % reservado a contingencias), repartidas entre aproximadamente 4 meses y medio, del 12 de setiembre de 2017 al 26 de enero de 2018.

Aun así, las reuniones con el profesor en relación al presente proyecto empezaron a mediados de Julio de 2017 con la propuesta del proyecto y la investigación de los primeros conceptos relacionados con el trabajo. Por otro lado, su fecha de finalización está prevista para la primera semana de enero, quedando por hacer hacia esa fecha la memoria del proyecto.

La dedicación semanal del proyecto será distinta y dependerá del volumen de trabajo realizado en cada estadio del mismo.

La planificación ha sido estructurada de manera que todas las fases tengan una temporalidad suficiente para no requerir más de 25 horas de trabajo semanales.

8.2. Descripción de tareas y recursos

En este apartado se describen las tareas que se deben realizar, separadas por las distintas secciones del proyecto estipuladas por el módulo de gestión de proyectos de la Facultad así como se destacarán los diferentes roles que tomarán partido en cada una de ellas.

Las fechas de entregas han sido marcadas en la planificación en base a los objetivos a cumplir del proyecto que han sido mencionados con anterioridad.

Tal como se definió en el contexto y en el alcance del proyecto, se utiliza una metodología ágil de trabajo [12]. Por tanto, el objetivo de la planificación no es marcar unas tareas con el objetivo de realizarlas secuencialmente y/o de manera estricta.

En cada sprint, de dos semanas de duración, se definirán los objetivos a alcanzar para la siguiente iteración, adaptando el proceso a posibles desviaciones o problemas que puedan surgir.

Aún teniendo en cuenta las desviaciones y con el objetivo de evitar posibles problemas al dilatar en el tiempo los problemas u obstáculos, se procede a definir unas tareas genéricas que engloban un conjunto de objetivos y que sí pueden acotarse aproximadamente en el tiempo.

8.2.1. Fase inicial

Se realizará un estudio de búsqueda y recopilación de información sobre el estado del arte para evaluar posibles maneras de afrontar el problema, tecnologías a utilizar, planificación adecuada para el alcance definido y evaluación de riesgos y sus respectivas contingencias.
Principalmente:

1. Definición del alcance y contexto
2. Definición temporal
3. Gestión económica y sostenibilidad

8.2.2. Análisis y diseño

Una vez analizado el estado del arte del proyecto y las tecnologías en uso, se deberá estructurar una posible solución al problema planteado en el proyecto.

8.2.3. Configuración del entorno

Estructurada la solución a implementar, se configurar del entorno de trabajo y las herramientas necesarias para ello.

8.2.4. Análisis de artículos e investigaciones

Se realizará una búsqueda, estudio y análisis de artículos e investigaciones para identificar los diferentes escenarios de aplicación del método KDE.

8.2.5. Visor de resultados

Aún cuando el sistema esté definido, el visor de resultados puede ser tratado como un elemento independiente al problema, pues únicamente será utilizado como una herramienta de análisis a la hora de interpretar los resultados.

8.2.6. Implementación de los algoritmos

Fase primordial en el trabajo dado que en ella se desarrollará la solución a los dos problemas planteados en la formulación del problema (Apartado 1.3 del documento) así como de los algoritmos de análisis de los resultados y de los datasets.

8.2.7. Análisis de resultados

Fase que analizará los resultados obtenidos de KDE con el objetivo de encontrar si existen relaciones directas entre los parámetros de entrada del algoritmo y un control específico de sus desviaciones.

8.2.8. Fase final

En la fase final se presentará la memoria del proyecto que incluirá las implementaciones realizadas, los resultados obtenidos y los análisis extraídos relevantes en el transcurso del proyecto y las desviaciones que se hayan podido producir.

Se añadirán anexos con la documentación del código así como una guía de uso del visor utilizado.

8.3. Duración total esperada

Tarea	Duración aproximada
Hito inicial	80h
Análisis y diseño	60h
Configuración del entorno	10h
Análisis de artículos e investigaciones	100h
Visor de resultados	20h
Implementación de los algoritmos	150h
Análisis de resultados	20h
Hito final	30h
Total	470h

Tabla 1: Duración total esperada

8.4. Diagrama de Gantt

8.4.1. Planificación inicial

	Nombre	Inicio	Fin
1	**Hito inicial**	**12/09/2017**	**23/10/2017**
2	Definición del alcance y contextualización	12/09/2017	25/09/2017
3	Planificación temporal	26/09/2017	02/10/2017
4	Gestión económica y sostenibilidad	03/10/2017	09/10/2017
5	Presentación preliminar	10/10/2017	12/10/2017
6	Bibliografia	13/10/2017	17/10/2017
7	Revisión de las competencias	18/10/2017	19/10/2017
8	Documentación presentación final	20/10/2017	23/10/2017
9	**Análisis y diseño**	**03/10/2017**	**13/10/2017**
10	Análisis	03/10/2017	06/10/2017
11	Diseño	09/10/2017	13/10/2017
12	**Configuración del entorno**	**16/10/2017**	**16/10/2017**
13	Editores y entornos de trabajo actualizados	16/10/2017	16/10/2017
14	Búsqueda de librerías para visualizar KDE	16/10/2017	16/10/2017
15	Descarga de la documentación de CGAL	16/10/2017	16/10/2017
16	**Análisis de artículos e investigaciones**	**17/10/2017**	**17/11/2017**
17	Búsqueda de artículos	17/10/2017	27/10/2017
18	Identificación de los parámetros utilizados	30/10/2017	31/10/2017
19	Obtención de los datos originales	30/10/2017	07/11/2017
20	Tratamiento de datos para unificar el input	08/11/2017	13/11/2017
21	Análisis de complejidad de los datos	14/11/2017	17/11/2017
22	Generación de datos aleatorios	23/10/2017	31/10/2017
23	**Visor de resultados**	**20/11/2017**	**28/11/2017**
24	Documentación de las librerías utilizadas	20/11/2017	20/11/2017
25	Diseño del visor	21/11/2017	22/11/2017
26	Implementación del visor	23/11/2017	27/11/2017
27	Pruebas de visualización	28/11/2017	28/11/2017
28	**Implementación de los algoritmos**	**20/11/2017**	**05/01/2018**
29	Diseño	20/11/2017	24/11/2017
30	Brute-force	27/11/2017	30/11/2017
31	Basado en Voronoi	01/12/2017	12/12/2017
32	Problema de conteo	13/12/2017	20/12/2017
33	Visualización de los resultados	21/12/2017	22/12/2017
34	Pruebas y validación	22/12/2017	05/01/2018
35	**Análisis de resultados**	**05/01/2018**	**17/01/2018**
36	Resultados obtenidos con los datos disponibles	05/01/2018	09/01/2018
37	Comparación entre ejecuciones experimentales y teóricas	09/01/2018	11/01/2018
38	Comparación de la relación de parámetros en las ejecuciones	11/01/2018	16/01/2018
39	Conclusiones	16/01/2018	17/01/2018
40	**Hito final**	**18/01/2018**	**26/01/2018**
41	Finalización del proyecto	18/01/2018	18/01/2018
42	Redacción de la memoria	19/01/2018	22/01/2018
43	Presentación	22/01/2018	26/01/2018

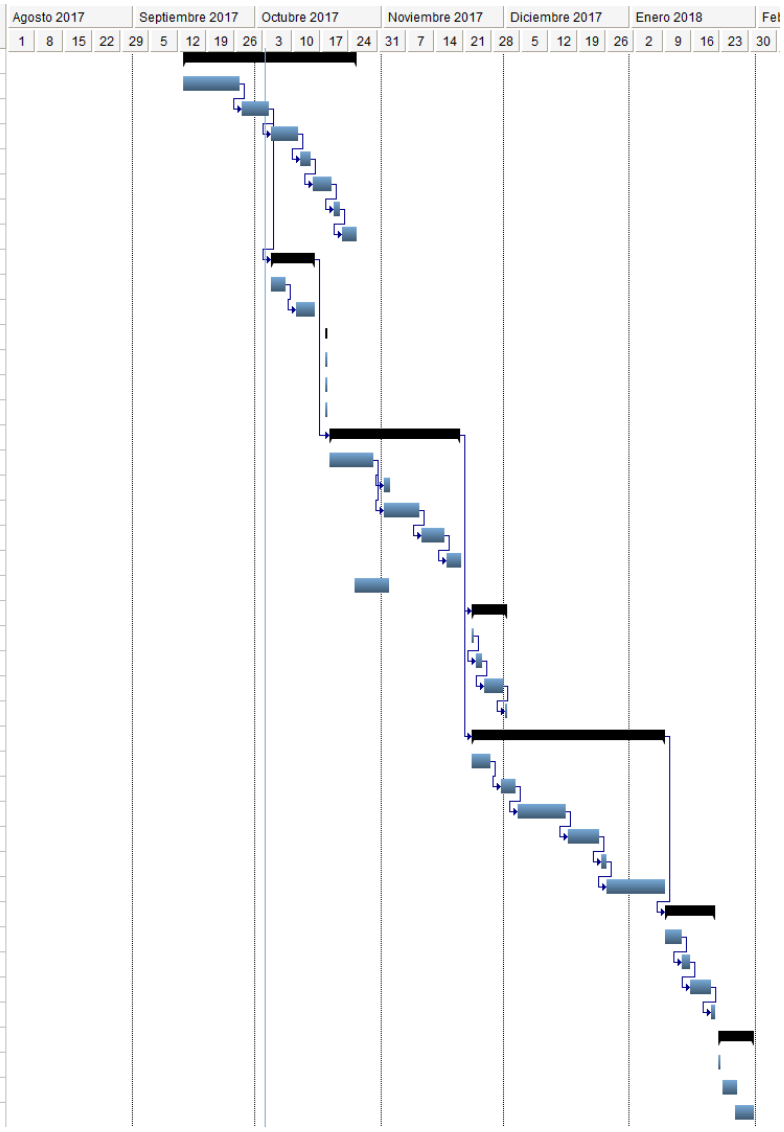


Figura 24: Diagrama de Gantt generado con [18]

Tareas propias del Jefe de Proyecto:

- Hito inicial
- Reuniones cada dos semanas
- Control de la planificación temporal y económica (riesgo por desviaciones)
- Hito final

Tareas propias del Full Stack Developer:

- Análisis y diseño
- Configuración del entorno
- Análisis de artículos e investigaciones
- Visualización de resultados
- Implementación de los algoritmos (riesgo por desviaciones)
- Análisis de los resultados

Tareas propias del Beta Tester:

- Configuración del entorno
- Visualización de resultados
- Test de los algoritmos
- Análisis de los resultados

8.4.2. Posibles desviaciones

Como ha sido comentado con anterioridad, en la realización del proyecto pueden surgir desviaciones respecto a la planificación original.

En caso de producirse y siguiendo los procedimientos ágiles, se analizarán sus consecuencias y se gestionarán en consecuencia para minimizar el impacto sobre el proyecto.

Si el desarrollo se ve afectado por desviaciones no planificadas, sean por errores de programación o imprevistos surgidos, se reducirá y simplificará de manera proporcional el aspecto de la visualización de los resultados, dado que este será algo meramente informativo e intuitivo para los lectores, pues al realizar juegos de prueba y tests sobre los resultados ya podrán ser verificados los resultados obtenidos.

En cuanto a las tareas, las horas fijadas para cada una únicamente sirven a modo orientativo, pudiéndose paralelizar tareas de desarrollo que sean independientes entre ellas.

Al final de proyecto se podrá hacer un seguimiento de la planificación resultante ejecutada a lo largo del proyecto, puesto que se habrá registrado en las actas de reunión de las diferentes iteraciones realizadas.

8.4.3. Desviaciones producidas

Cambios en la estructura y software del proyecto
En un inicio se planificó una estructura de comunicación entre los diferentes componentes en la que existía como intermediario un servidor NodeJS.

El servidor estaba como intermediario entre los diferentes módulos y se utilizaba a modo de API server. La principal desviación y, por tanto cambio en el modelo de estructura producida, apareció al trabajar con cantidades significativas de datos reales en las que el servidor no era capaz de transmitir de punto a punto la información e incrementaba de forma significativa el tiempo de ejecución.

Unificación de datos reales a un sistema que el software pueda interpretar de forma correcta
Dado que los datos reales obtenidos pertenecen al UK Data Service [16], estos utilizan un sistema de posicionamiento diferente al convencional para ubicaciones (latitud y longitud), se estableció un sistema para unificar todos los posibles datasets y de forma adecuada, transformarlos a dicho sistema.

Aún con las desviaciones producidas, el diagrama de Gantt siguió estando en buena temporización dado que se adelantaron e intensificaron las tareas de programación en fechas anteriores. Por este motivo, el diagrama no fue modificado y se continuaron las tareas previstas.

Cambios de tecnología a la hora de visualizar los datos

Si bien es cierto que en un inicio se tuvieron en cuenta herramientas como Matlab, se prefirió investigar otras de libre uso como la librería gráfica en lenguaje javascript llamada D3 (d3.js) [14].

Más adelante a la hora de hacer pruebas más allá de la representación del KDE, empezaron a surgir errores y discrepancias entre datos y se optó por abandonar dicha opción y enfocar la visualización de datos a través de Matlab aprovechando la licencia de estudiante de la que dispone el autor del proyecto.

8.5. Recursos materiales y software

Para la realización del proyecto, se dispone de diferentes herramientas de software así como un hardware específico.

- Portátil Asus R510J (Hardware)
- TexMaker [19](Software)
- Sublime Text [20](Software)
- Matlab R2017b [13](Software)

8.6. Integración de conocimientos

Para la realización del proyecto, se han aprovechado los conocimientos adquiridos de las diferentes asignaturas que se listan a continuación:

- **Geometria computacional, GEOC:** Conceptos básicos que me han permitido interesarme por el tema del TFG y entender de forma matemática todos los aspectos relacionados con los objetivos a cumplir.
- **Algorítmica, A:** Capacidad para analizar de forma correcta y objetiva los algoritmos propuestos e implementados y realizar, si es necesario, correcciones y optimizaciones a nivel de código.
- **Lógica en la informática, LI:** Capacidad para analizar de forma correcta y objetiva los algoritmos desde la perspectiva lógica.
- **Lenguajes de programación, LP:** Aporta la capacidad al desarrollador de trabajar con múltiples lenguajes de programación, esenciales para la implementación de los algoritmos, capacitándolo para abstraer y aprender cada uno de estos de forma ágil.
- **Gestión de Proyectos de Software, GPS:** Por los conocimientos adquiridos en esta asignatura, el desarrollador ha tenido los conceptos ya definidos para poder trabajar con la metodología Scrum.

9. Gestión económica

Con el objetivo principal de desarrollar el proyecto, se están siguiendo todos los procesos definidos y detallados en los apartados anteriores 3 8 que implican, directa o indirectamente, costes. Así, en este documento se procederá a proporcionar la estimación del coste del proyecto en relación a los recursos humanos, el hardware, el software (licencias) y todos aquellos costes indirectos que puedan añadirse y que surgen o son derivados de las tareas que han sido definidas en el diagrama de Gantt de la figura 24.

Dada la importancia del presupuesto en un proyecto, se irán actualizando las previsiones conforme se realicen las diferentes tareas definidas y por tanto, el presupuesto final será un compendio de las horas reales que se han trabajado y el precio final.

9.1. Presupuesto de recursos humanos

Éste proyecto únicamente está siendo desarrollado por una persona que se encarga de todos los roles que fueron definidos con anterioridad:

- Jefe de proyecto
- Full Stack Developer
- Beta Tester

En la siguiente tabla se especifican las horas aproximadas que serán necesarias para realizar todas las tareas de cada rol y su valor económico correspondiente:

Rol	Horas	Precio por hora	Precio total
Jefe de proyecto	120h	50€/h	6000€
Full Stack Developer	290h	35€/h	10150€
Beta Tester	60h	30€/h	1800€
Total	470h	-	17950€

Tabla 2: Presupuesto de recursos humanos

Una vez realizado, se procederá a calcular el desvío que se ha producido con la siguiente fórmula:

*Desvío de mano de obra = (coste std - coste real) * consumo horas real (desvío en coste por tarifa)*

9.2. Presupuesto por fases

Para justificar las horas destinadas a cada uno de los roles y en base al diagrama de Gantt que se definió con anterioridad, la siguiente tabla tiene como objetivo desglosar el presupuesto de recursos humanos por cada una de las fases del proyecto:

Concepto	Coste planificado
Hito inicial	3300€
Análisis y diseño	1850€
Configuración del entorno	1600€
Análisis de artículos e investigaciones	2200€
Visor de resultados	800€
Implementación de los algoritmos	4200€
Análisis de resultados	1750€
Hito final	2250€
Total	17950€

Tabla 3: Presupuesto por fases

9.3. Presupuesto de hardware

Para poder llevar a cabo el proyecto es necesario utilizar software y, por consecuencia directa, utilizar hardware, sea para tareas de documentación, organización, implementación o pruebas.

En concreto, el único hardware que se utilizará será un ordenador portátil:

Producto	Precio	Unidades	Vida útil	Amortización
Asus R510JF 15.6”	768,50€	1	5 años	36,12€
Total	768,50€	-	-	36,12€

Tabla 4: Presupuesto de hardware

El coste de amortización es de 768,5€/5 años = 153,7€al año.

Contando que el ordenador tiene un uso de 8h al día, unos 250 días al año laborables 250*8 = 2000 horas. Por tanto, 153,7€/2000 horas = 0,076 €la hora.

Si se utiliza el ordenador durante las 470 horas del proyecto, el coste de amortización es de 36,12€.

9.4. Presupuesto de software

Para poder desarrollar el proyecto es necesario utilizar software, a continuación se detalla todo el software y licencias utilizados en el proyecto:

Producto	Precio	Unidades	Vida útil	Amortización
TexMaker	0,00€	1	3 años	0,00€
Sublime Text 3	0,00€	1	3 años	0,00€
CGAL Non-commercial	0,00€	1	3 años	0,00€
Matlab - Academic use, individual	500,00€	1	1 año	72,50€
Linux	0,00€	1	3 años	0,00€
Total	500,00€	-	-	72,50€

Tabla 5: Presupuesto de sofware

El coste de amortización es de 500€/1 año = 500€al año.

Contando que el software de Matlab tiene un uso de 8h al día, unos 250 días al año laborables, 250 días * 8 horas = 2000 horas. Por tanto, 500€/2000 horas = 0,25€/ hora.

Si se utiliza durante toda la fase de implementación, 290 horas, el coste de amortización es de 72,50€.

9.5. Gastos indirectos

En todo proyecto de informática aparecen gastos indirectos derivados del uso de electricidad, papel o Internet entre otros.

Producto	Precio	Relación coste	Coste
Electricidad	0,14767€/kW·h	70W·h * 470horas = 32,9 kW·h * 0,14767€	4,86€
Internet	30€/mes	30€/mes * 0,4 % de actividad * 5 meses	60€
Transporte	6,00€/viaje ida y vuelta	6,00€* reunión cada 2 semanas * 4,5 meses	54€
Total	-	-	118,86€

Tabla 6: Gastos indirectos

9.6. Presupuesto total

A todo lo descrito con anterioridad y para poder avanzar ante los posibles imprevistos, como prevención, se destinará un 5 % del total del presupuesto a contingencias que puedan ocasionarse, de tal forma que el presupuesto total quedará definido por:

Concepto	Coste aproximado
Recursos humanos	17950€
Hardware	36,12€
Software y licencias	72,5€
Costes indirectos	118,86€
Contingencia (5 %)	908,87€
Total	19086,35€

Tabla 7: Presupuesto total

Como consecuencia de utilizar en gran parte software disponible open source y de manera gratuita, la mayor carga de costes vienen definidos por los recursos humanos necesarios.

9.7. Control de desviaciones

El principal problema que puede aparecer es la desviación temporal en algunas de las tareas definidas en el proyecto.

Con el objetivo de minimizarlas, en caso de producirse, se utilizaría el diagrama de Gantt para reorganizar el tiempo y optimizar las tareas, descartando en caso de estricta necesidad aspectos como la visualización de los resultados y uniendo todos los recursos disponibles en la implementación del algoritmo y en el análisis de sus resultados.

10. Informe de sostenibilidad

10.1. Área social

El proyecto está definido en el entorno de la ingeniería informática, más concretamente en el ámbito de la investigación, la privacidad y la geometría computacional. Los estudios de datos geográficos son realizados constantemente y cada vez más son utilizados para extraer información de usuarios que indirectamente han proporcionado sus datos. Éste proyecto pretende pues, en el ámbito de los datos geométricos, ayudar a compensar la escasa privacidad que poseen los individuos en la red.

El proyecto está basado, como se ha comentado en documentos anteriores, en un artículo de investigación. El objetivo principal es la continuación de éste y el análisis empírico de los algoritmos que se describen en el artículo mencionado, así como de su implementación para automatizar el cálculo del mejor threshold posible en función de los datos introducidos.

10.2. Área ambiental

En el área ambiental las incidencias son mínimas, dado que íntegramente está en el entorno del software y la computación y los únicos gastos que pueden afectar al medio ambiente son algunos de los costes indirectos mencionados con anterioridad.

10.3. Puntuación de sostenibilidad

En la siguiente sección se analiza la dimensión ambiental, económica y social del proyecto.

En la siguiente tabla se pondera cada una de las secciones que abarcan el proyecto, del 1 al 10, y se evalúa la sostenibilidad sobre 30.

Económica	Social	Ambiental	Sostenibilidad
8	9	7	24/30

Tabla 8: Matriz de sostenibilidad

11. Conclusiones

Para realizar este proyecto, antes de desarrollarlo, se establecieron objetivos genéricos y específicos.

Salvo el análisis entre optimizaciones de búsqueda del threshold máximo, que por motivos relacionados la falta de tiempo, no se han podido implementar más allá del algoritmo de Brute Force, se puede afirmar que se han alcanzado todos los otros satisfactoriamente.

En cuanto a la búsqueda de una relación directa entre los parámetros de entrada para la técnica de geomasking y los resultados obtenidos de KDE, se puede intuir que sí es posible que existan relaciones directas y, por tanto, se pueda decidir a priori qué porcentaje de resultados se desea sacrificar en pro de la privacidad de los usuarios que los proveen.

Sobre la GUI implementada, es necesario recordar que no era uno de los objetivos principales del proyecto, pero dado que era la visualización de todo el trabajo que ha existido en el proyecto, la agilidad que ha supuesto a la hora de evaluar los resultados y la utilidad que ha tenido para el desarrollador, ha cumplido todas las expectativas esperadas.

Gracias a este proyecto he podido aplicar los conocimientos que he adquirido a lo largo del grado. He añadido nuevos conocimientos relacionados con la materia del tema en el que se engloba el proyecto y he puesto en práctica metodologías de trabajo ágiles que me han permitido organizarme y enfrentar retos de planificación y técnicos.

11.1. Trabajo futuro

Para el algorito de fuerza bruta:

- Mejora de la implementación de la búsqueda de δ para alcanzar una complejidad temporal de $O(nm)$ con implementaciones de inserción y búsqueda como *Linear time median algorithm* [32]
- Agregación de nuevos datasets
- Añadir parámetros como el nivel de aislamiento entre los diferentes clústers de puntos en cada uno de los datasets o la relación de variación entre la densidad de puntos por área.

Apéndices

Búsqueda de datasets

Información extraída de la búsqueda de información de datasets:

- *Kernel density estimation and K-means clustering to profile road accident hotspots*
  Size of grid cell: 100m
  Radius/Bandwidth: 2*size of grid cell: 200m
  Ref: doi:10.1016/j.aap.2008.12.014, Pg: 3
  Data: *https://discover.ukdataservice.ac.uk/series/?sn=2000045*

- *Anomaly detection in sea traffic - a comparison of the Gaussian Mixture Model and the Kernel Density Estimator*
  Size of grid cell: -
  Western Coast of Sweden, Grid 12x24 cells, 22 km of length
  Gothenburg Grid 6x4 cells, 2 km of length
  Radius/Bandwidth: -
  Ref: -, Pg: 4
  Data: -

- *Identification of hazardous road locations of traffic accidents by means of kernel density estimation and cluster significance evaluation*
  Size of grid cell: 200m
  Radius/Bandwidth: 400m
  Ref: doi:10.1016/j.aap.2013.03.003, Pg: 2
  Data: -

- *Network Density and the Delimitation of Urban Areas* **Size of grid cell:** 100m
  Radius/Bandwidth: 500m
  Ref: doi:10.1111/1467-9671.00139 Pg: 6, 9
  Data: -

- *Using kernel density estimation to assess the spatial pattern of road density and its impact on landscape fragmentation*
  Size of grid cell: -
  Radius/Bandwidth: 4 km
  Ref: doi:10.1111/1467-9671.00139 Pg: 6
  Data: -

- *Automated Footprint Generation from Geotags with Kernel Density Estimation and Support Vector Machines*
  Size of grid cell: -
  Radius/Bandwidth: -
  Ref: doi:10.1080/13875860903118307 Pg: -
  Data: -

- *Examining the influence of interpolation method, grid cell size, and bandwidth on crime forecasting*
  Size of grid cell: 250x150 columns bounding rectangle
  Radius/Bandwidth: 1/4, 1/2 and 1 mile
  Ref: doi:10.1108/PIJPSM-04-2013-0039 Pg: 9 - 10
  Data: Crimes recorded by Arlington (Texas) Police Department, incidents matched, and match rates

- *GIS-based method for detecting high-crash-risk road segments using network kernel density estimation*
  Size of grid cell: -
  Radius/Bandwidth: 1km
  Ref: doi:10.1080/10095020.2013.766396 Pg: -
  Data: -

- *Use of kernel density estimation and maximum curvature to set Marine Protected Area boundaries: Identifying a Special Protection Area for wintering red-throated divers in the UK*
  Size of grid cell: 1km
  Radius/Bandwidth: -
  Ref: doi:10.1016/j.biocon.2011.12.033 Pg: 17
  Data: -

- *A Network Based Kernel Density Estimator Applied to Barcelona Economic Activities*
  Size of grid cell: 10m
  Radius/Bandwidth: 100m
  Ref: - Pg: 13
  Data: -

- *Kernel density estimation as a technique for assessing availability of health services in Nicaragua*
  Size of grid cell: 1 km
  Radius/Bandwidth: -
  Ref: 10.1007/s10742-007-0022-7 Pg: 6
  Data: -

Capturas de la GUI implementada

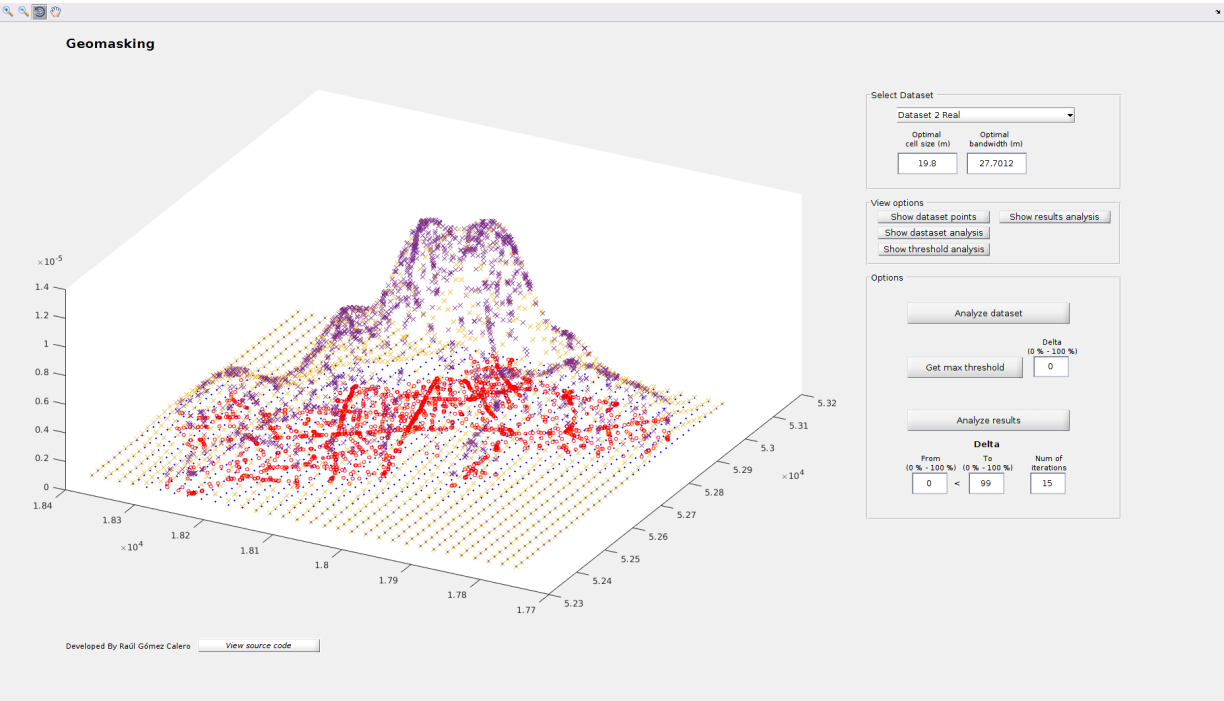


Figura 25: GUI pantalla de visualización del KDE con los dos métodos Grid Y Kernel y el menú lateral

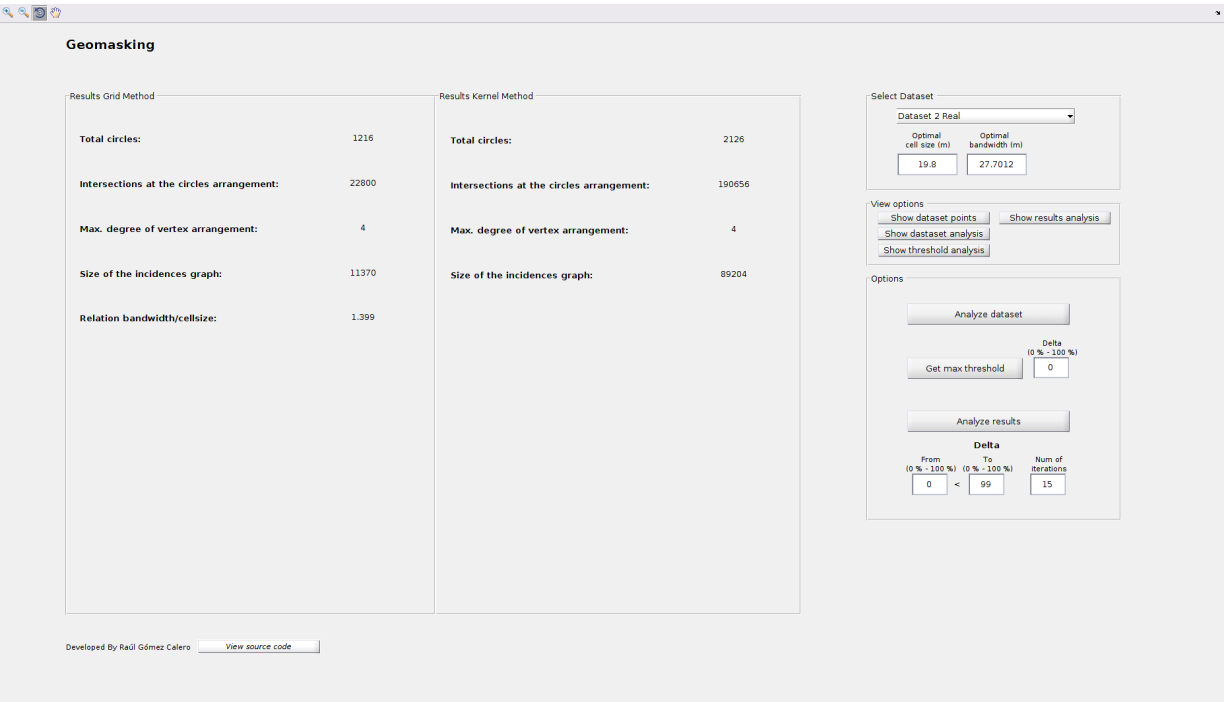


Figura 26: GUI pantalla de análisis del dataset seleccionado

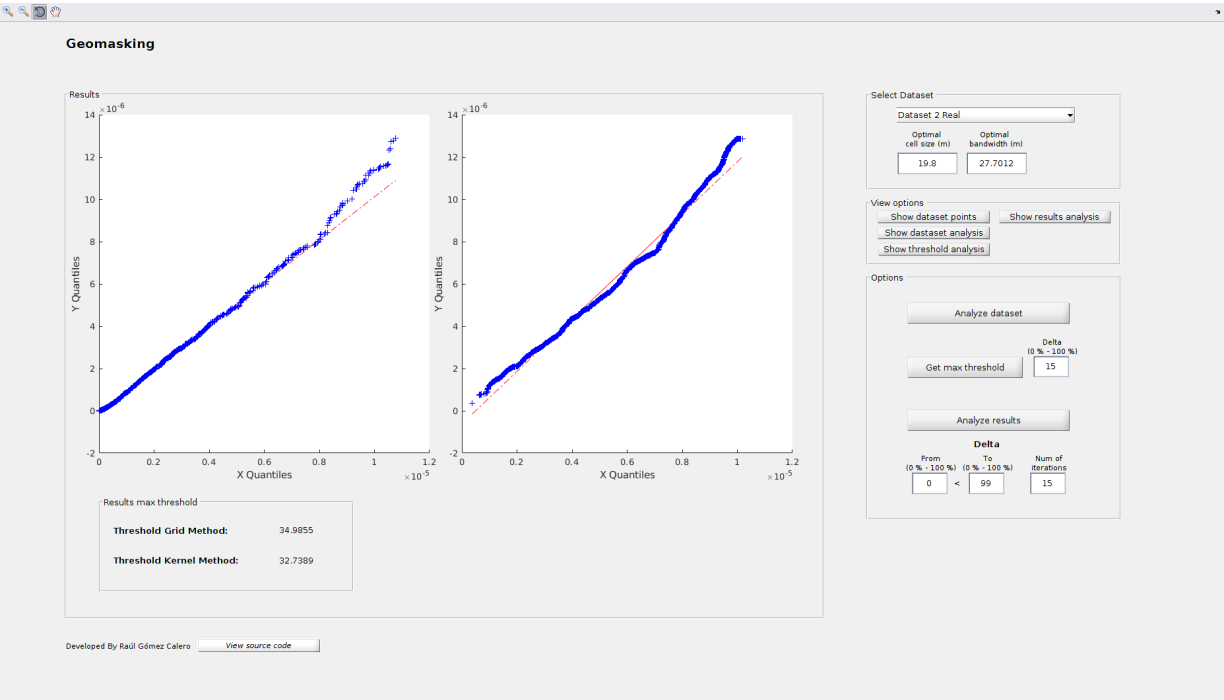


Figura 27: GUI pantalla del análisis del threshold máximo

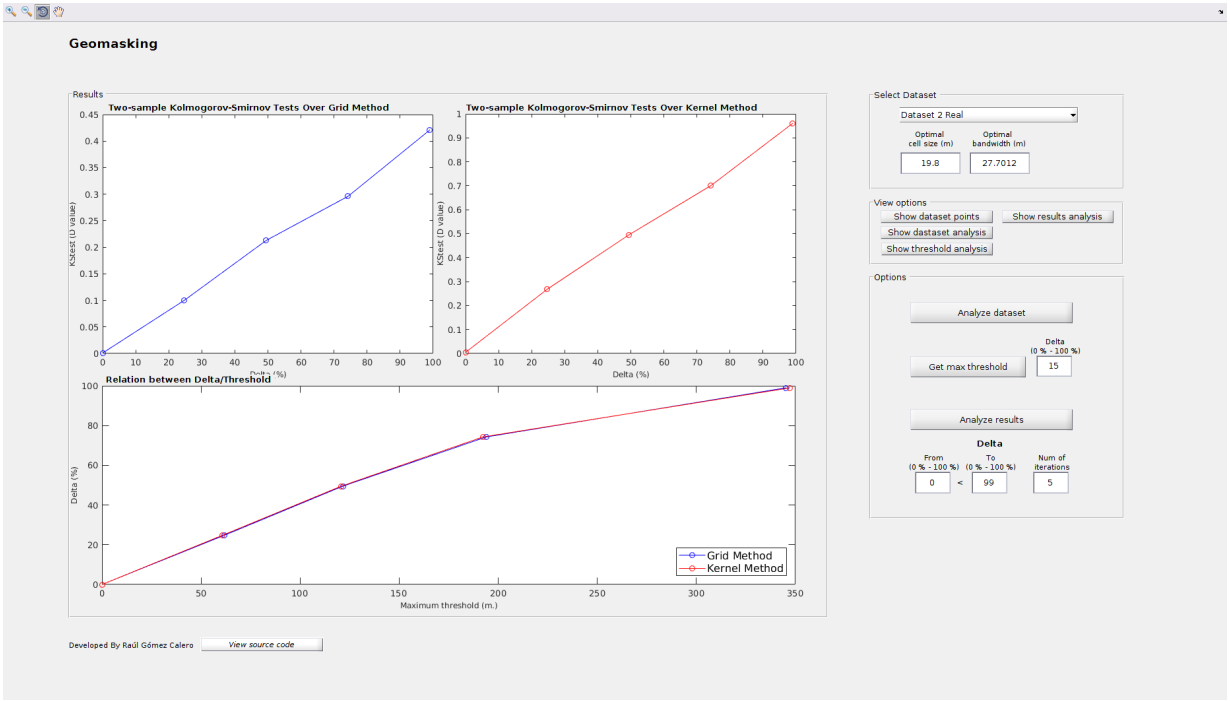


Figura 28: GUI pantalla de los resultados del test de Kolmogorov-Smirnov

Código implementado

gui.m

```
function varargout = test_gui(varargin)
% TEST_GUI MATLAB code for test_gui.fig
%      TEST_GUI, by itself, creates a new TEST_GUI or raises the existing
%      singleton*.
%
%      H = TEST_GUI returns the handle to a new TEST_GUI or the handle to
%      the existing singleton*.
%
%      TEST_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TEST_GUI.M with the given input arguments.
%
%      TEST_GUI('Property','Value',...) creates a new TEST_GUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before test_gui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to test_gui_Fcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help test_gui

% Last Modified by GUIDE v2.5 04-Jan-2018 22:07:09

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @test_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @test_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
```

```
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% Initialize paths
addpath(genpath('../'));

% --- Executes just before test_gui is made visible.
function test_gui_OpeningFcn(hObject, ~, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to test_gui (see VARARGIN)

% If real datasets are not generated, generate them.
% if ~exist('dataset1real.tab', 'file') || ...
%    ~exist('dataset2real.tab', 'file')
%     generateRealDatasets;
% end

% Default dataset selected and other variables
axes(handles.axes1);
handles.current_dataset = 'a1';
set(handles.delta, 'String', 0.0);
set(handles.select_dataset, 'Value', 1);

set(handles.from, 'String', 0.0);
set(handles.to, 'String', 99.0);
set(handles.iterations, 'String', 15);
```

```
handles.current_data = dlmread("../datafiles/"+handles.current_dataset+".tab");
scatter(handles.axes1, handles.current_data(:, 1), handles.current_data(:, 2), 15, 'r');

% Choose default command line output for test_gui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

f = @show_view_1;
f(handles);

set_cell_size(hObject, handles);

set_bandwidth(hObject, handles);


% --- Outputs from this function are returned to the command line.
function varargout = test_gui_OutputFcn(~, ~, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on selection change in select_dataset.
function select_dataset_Callback(hObject, ~, handles) %#ok<*DEFNU>
% hObject    handle to select_dataset (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

axes(handles.axes1);
```

```
% Determine the selected data set.
str = get(hObject, 'String');
val = get(hObject,'Value');

% Set current data to the selected data set.
switch str{val}
    case 'A1'
        handles.current_dataset = 'a1';
    case 'A2'
        handles.current_dataset = 'a2';
    case 'A3'
        handles.current_dataset = 'a3';
    case 'S1'
        handles.current_dataset = 's1';
    case 'S2'
        handles.current_dataset = 's2';
    case 'S3'
        handles.current_dataset = 's3';
    case 'S4'
        handles.current_dataset = 's4';
    case 'Unbalance'
        handles.current_dataset = 'unbalance';
    case 'Test data'
        handles.current_dataset = 'test_data';
    case 'Dataset 1 Real'
        handles.current_dataset = 'dataset1real';
    case 'Dataset 2 Real'
        handles.current_dataset = 'dataset2real';
end

handles.current_data = dlmread("../datafiles/"+handles.current_dataset+".tab");
scatter(handles.axes1, handles.current_data(:, 1), handles.current_data(:, 2), 15, 'r');

f = @show_view_1;
f(handles);

set(handles.max_threshold_grid, 'String', '-');
```

```
set(handles.max_threshold_kernel, 'String', '-');
set(handles.text39, 'String', '-');
set(handles.text38, 'String', '-');
set(handles.text26, 'String', '-');
set(handles.text27, 'String', '-');
set(handles.text47, 'String', '-');
set(handles.text43, 'String', '-');
set(handles.text44, 'String', '-');
set(handles.text28, 'String', '-');
set(handles.text45, 'String', '-');

% Save handles structure
guidata(hObject, handles)

set_cell_size(hObject, handles);

set_bandwidth(hObject, handles);


% --- Executes during object creation, after setting all properties.
function select_dataset_CreateFcn(~, ~, ~)

% --- Executes on button press in analyze_dataset.
function analyze_dataset_Callback(~, ~, handles)
    % Set visibility of components
    f = @show_view_2;
    f(handles);

    % CGAL execution
    wb = waitbar(0.7, 'CGAL execution in progress ...');
    command = "../c++/analysis.x "+ ...
        str2double(get(handles.cellSize, 'String'))+ ' ' + ...
        str2double(get(handles.bandwidth, 'String')) + ' ' + ...
        handles.current_dataset;

    [~, cmdout] = system(command);
    handles.result = str2num(cmdout);       %#ok<*ST2NM>
```

```
    % Show variables
    set(handles.text39, 'String', num2str(handles.result(1)));
    set(handles.text38, 'String', num2str(handles.result(2)));
    set(handles.text26, 'String', num2str(handles.result(3)));
    set(handles.text27, 'String', num2str(handles.result(4)));
    set(handles.text47, 'String', num2str(handles.result(5)));
    set(handles.text43, 'String', num2str(handles.result(6)));
    set(handles.text44, 'String', num2str(handles.result(7)));
    set(handles.text28, 'String', num2str(handles.result(8)));
    set(handles.text45, 'String', num2str(handles.result(9)));
    close(wb);

function thresholds = compute_threshold(handles, initdelta, enddelta, iterations)
    wb = waitbar(0.7, 'CGAL execution in progress ...');
    command = "../c++/compute.x "+ ...
        str2double(get(handles.cellSize, 'String'))+ ' ' + ...
        str2double(get(handles.bandwidth, 'String'))+ ' ' + ...
        handles.current_dataset + ' ' + initdelta + ' ' + ...
        enddelta + ' ' + iterations;
    [~, cmdout] = system(command);
    thresholds = str2num(cmdout);       %#ok<*ST2NM>

    close(wb);

% --- Executes on button press in max_threshold.
function max_threshold_Callback(~, ~, handles)
    f = @show_view_3;
    f(handles);
    delta = str2double(get(handles.delta, 'String'));
    thresholds = compute_threshold(handles, delta, delta, 1);
    set(handles.max_threshold_grid, 'String', num2str(thresholds(1)));
    set(handles.max_threshold_kernel, 'String', num2str(thresholds(2)));

    fdg = ksdensity(handles.current_data, handles.xi);
    fdk = ksdensity(handles.current_data, handles.current_data);
    data_grid = zeros(size(fdg, 1), 30);
```

```
    data_kernel = zeros(size(fdk, 1), 30);
    for i=1:30
        rdata1 = zeros(size(handles.current_data));
        rdata2 = zeros(size(handles.current_data));
        for j = 1:size(handles.current_data, 1)
            [rdata1(j, 1), rdata1(j, 2)] = randomizePoint( ...
                handles.current_data(j, 1), ...
                handles.current_data(j, 2), ...
                thresholds(1));
            [rdata2(j, 1), rdata2(j, 2)] = randomizePoint( ...
                handles.current_data(j, 1), ...
                handles.current_data(j, 2), ...
                thresholds(2));
        end
        t1 = ksdensity(rdata1, handles.xi);
        t2 = ksdensity(rdata2, rdata2);
        data_grid(:, i) = t1(:, 1);
        data_kernel(:, i) = t2(:, 1);
    end

    axes(handles.axes3);
    qqplot(mean(data_grid, 2), fdg);
    axes(handles.axes7);
    qqplot(mean(data_kernel, 2), fdk);

% --- Executes on button press in analyze_results.
function analyze_results_Callback(~, ~, handles)
    f = @show_view_4;
    f(handles);
    from = str2double(get(handles.from, 'String'));
    to = str2double(get(handles.to, 'String'));
    iterations = str2double(get(handles.iterations, 'String'));

    thresholds = compute_threshold(handles, from, to, iterations);
    deltas = linspace(from, to, iterations);
    grid_thresholds = thresholds(1:2:end);
    kernel_thresholds = thresholds(2:2:end);
```

```
plot(handles.axes10, grid_thresholds, deltas, 'b-o', kernel_thresholds, ...
 deltas, 'r-o');
xlabel(handles.axes10, 'Maximum threshold (m.)', 'FontSize', 9);
ylabel(handles.axes10, 'Delta (%)', 'FontSize', 9);
lgd = legend(handles.axes10, 'Grid Method','Kernel Method','Location','southeast');
lgd.FontSize = 12;

plot_data_grid = zeros(iterations, 1);
plot_data_kernel = zeros(iterations, 1);
iterator = 1;

fdg = ksdensity(handles.current_data, handles.xi);
fdk = ksdensity(handles.current_data, handles.current_data);

for i = 1:2:size(thresholds)
    rdata1 = zeros(size(handles.current_data));
    rdata2 = zeros(size(handles.current_data));
    for j = 1:size(handles.current_data, 1)
        [rdata1(j, 1), rdata1(j, 2)] = randomizePoint( ...
            handles.current_data(j, 1), ...
            handles.current_data(j, 2), ...
            thresholds(i));
        [rdata2(j, 1), rdata2(j, 2)] = randomizePoint( ...
            handles.current_data(j, 1), ...
            handles.current_data(j, 2), ...
            thresholds(i+1));
    end
    fs1 = ksdensity(rdata1, handles.xi);
    fs2 = ksdensity(rdata2, rdata2);
    [~,~,ks2statg] = kstest2(fdg, fs1);
    [~,~,ks2statk] = kstest2(fdk, fs2);
    plot_data_grid(iterator) = ks2statg;
    plot_data_kernel(iterator) = ks2statk;
    iterator = iterator + 1;
end
axes(handles.axes8);
```

```
    plot(deltas, plot_data_grid, 'b-o');
    xlabel('Delta (%)', 'FontSize', 9) % x-axis label
    ylabel('KStest (D value)', 'FontSize', 9) % y-axis label
    axes(handles.axes9);
    plot(deltas,plot_data_kernel, 'r-o');
    xlabel('Delta (%)', 'FontSize', 9) % x-axis label
    ylabel('KStest (D value)', 'FontSize', 9) % y-axis label
    disp(plot_data_grid);
    disp(plot_data_kernel);

% AUX METHODS
function [x, y] = randomizePoint(x1,y1,rc)
    a=2*pi*rand;
%     r=sqrt(rand);
    x=(rc)*cos(a)+x1;
    y=(rc)*sin(a)+y1;
end

function show_view_1(handles)
    set(handles.uipanel4,'visible','off')
    set(handles.uipanel5,'visible','off')
    set(handles.axes1,'visible','on')
    set(get(handles.axes1,'children'),'visible','on')
    set(handles.uipanel10,'visible','off')
    set(handles.axes3,'visible','off')
    set(get(handles.axes3,'children'),'visible','off')
    set(handles.axes7,'visible','off')
    set(get(handles.axes7,'children'),'visible','off')
    set(handles.uipanel11,'visible','off')
    set(handles.axes8,'visible','off')
    set(get(handles.axes8,'children'),'visible','off')
    set(handles.axes9,'visible','off')
    set(get(handles.axes9,'children'),'visible','off')

function show_view_2(handles)
    set(handles.axes1,'visible','off')
    set(get(handles.axes1,'children'),'visible','off')
```

```
    set(handles.uipanel4,'visible','on')
    set(handles.uipanel5,'visible','on')
    set(handles.uipanel10,'visible','off')
    set(handles.axes3,'visible','off')
    set(get(handles.axes3,'children'),'visible','off')
    set(handles.axes7,'visible','off')
    set(get(handles.axes7,'children'),'visible','off')
    set(handles.uipanel11,'visible','off')
    set(handles.axes8,'visible','off')
    set(get(handles.axes8,'children'),'visible','off')
    set(handles.axes9,'visible','off')
    set(get(handles.axes9,'children'),'visible','off')

function show_view_3(handles)
    set(handles.axes1,'visible','off')
    set(get(handles.axes1,'children'),'visible','off')
    set(handles.uipanel4,'visible','off')
    set(handles.uipanel5,'visible','off')
    set(handles.uipanel10,'visible','on')
    set(handles.axes3,'visible','on')
    set(get(handles.axes3,'children'),'visible','on')
    set(handles.axes7,'visible','on')
    set(get(handles.axes7,'children'),'visible','on')
    set(handles.uipanel11,'visible','off')
    set(handles.axes8,'visible','off')
    set(get(handles.axes8,'children'),'visible','off')
    set(handles.axes9,'visible','off')
    set(get(handles.axes9,'children'),'visible','off')

function show_view_4(handles)
    set(handles.axes1,'visible','off')
    set(get(handles.axes1,'children'),'visible','off')
    set(handles.uipanel4,'visible','off')
    set(handles.uipanel5,'visible','off')
    set(handles.uipanel10,'visible','off')
    set(handles.axes3,'visible','off')
    set(get(handles.axes3,'children'),'visible','off')
```

```
    set(handles.axes7,'visible','off')
    set(get(handles.axes7,'children'),'visible','off')
    set(handles.uipanel11,'visible','on')
    set(handles.axes8,'visible','on')
    set(get(handles.axes8,'children'),'visible','on')
    set(handles.axes9,'visible','on')
    set(get(handles.axes9,'children'),'visible','on')

% Bounding box of the dataset
function [startX, endX, startY, endY] = get_bounding_box(current_data)
    startX = min(current_data(:, 1));
    endX = max(current_data(:, 1));
    startY = min(current_data(:, 2));
    endY = max(current_data(:, 2));

function set_cell_size(hObject, handles)
    % Based on https://www.mathworks.com/help/stats/ksdensity.html
    % 900 cells at grid for bivariate data --> grid 30x30
    [startX, endX, startY, endY] = get_bounding_box(handles.current_data);

    min_side = min(endX - startX, endY - startY);
    set(handles.cellSize, 'String', min_side/30.0);

    % Update handles structure
    guidata(hObject, handles);

function set_bandwidth(hObject, handles)
    % Based on https://www.mathworks.com/help/stats/ksdensity.html
    [startX, endX, startY, endY] = get_bounding_box(handles.current_data);

    if endX - startX <= endY - startY
        bw_side = 1;
    else
        bw_side = 2;
    end

    min_side = min(endX - startX, endY - startY);
```

```
    cellSize = min_side/30.0;
    midCell = cellSize/2;

    gridx1 = startX-midCell:cellSize:endX+midCell;
    gridx2 = startY-midCell:cellSize:endY+midCell;
    [x1,x2] = meshgrid(gridx1, gridx2);
    x1 = x1(:);
    x2 = x2(:);
    handles.xi = [x1 x2];
    hold on;
    scatter(handles.xi(:, 1), handles.xi(:, 2), 1, 'b', '*');
    ksdensity(handles.current_data, handles.xi, 'PlotFcn', 'plot3');
    ksdensity(handles.current_data, handles.current_data, 'PlotFcn', 'plot3');
    hold off

    [~,~,bw] = ksdensity(handles.current_data, handles.xi);

    set(handles.bandwidth, 'String', bw(bw_side));

    % Update handles structure
    guidata(hObject, handles);

% Cell size parameter
function cellSize_Callback(~, ~, ~)

function cellSize_CreateFcn(hObject, ~, ~)
    if ispc && isequal(get(hObject,'BackgroundColor'), ...
     get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

% Bandwidth parameter
function bandwidth_Callback(~, ~, ~)

function bandwidth_CreateFcn(hObject, ~, ~)
    if ispc && isequal(get(hObject,'BackgroundColor'), ...
     get(0,'defaultUicontrolBackgroundColor'))
```

```
        set(hObject,'BackgroundColor','white');
    end

% Delta parameter
function delta_Callback(~, ~, ~)

function delta_CreateFcn(hObject, ~, ~)
    if ispc && isequal(get(hObject,'BackgroundColor'), ...
     get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

% Initial delta
function from_Callback(~, ~, ~)

function from_CreateFcn(hObject, ~, ~)
    if ispc && isequal(get(hObject,'BackgroundColor'), ...
     get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

% Final delta
function to_Callback(~, ~, ~)

function to_CreateFcn(hObject, ~, ~)
    if ispc && isequal(get(hObject,'BackgroundColor'), ...
     get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

% # of iterations for delta results
function iterations_Callback(~, ~, ~)

function iterations_CreateFcn(hObject, ~, ~)
if ispc && isequal(get(hObject,'BackgroundColor'), ...
 get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end

% --- Executes on button press in show_view_1.
function show_view_1_Callback(~, ~, handles)
    show_view_1(handles);

% --- Executes on button press in show_view_2.
function show_view_2_Callback(~, ~, handles)
    show_view_2(handles);

% --- Executes on button press in show_view_3.
function show_view_3_Callback(~, ~, handles)
    show_view_3(handles);

% --- Executes on button press in show_view_4.
function show_view_4_Callback(~, ~, handles)
    show_view_4(handles);

% View source code
function view_code_Callback(~, ~, ~)
    web("https://github.com/shenrgc/geomasking", '-browser');

function generateRealDatasets
    % Load UK Data
    load('ukdata.mat');

    % Dataset 1
    indexes = (A1_10 < 32) | (A1_10 == 57) | (A1_10 == 570);
    indexes = (indexes) & (A1_2 ~= 1);
    dataset = [A10(indexes), A11(indexes)];
    dlmwrite('../datafiles/dataset1real.tab', dataset,'delimiter','\t');

    % Dataset 2
    indexes = (A1_10 == 1);
    dataset = [A10(indexes), A11(indexes)];
    dlmwrite('../datafiles/dataset2real.tab', dataset,'delimiter','\t');
```

```
    % Dataset 3
    indexes = (A1_10 < 32) | (A1_10 == 57) | (A1_10 == 570);
    indexes = (indexes) & (A3 == 3);
    dataset = [A10(indexes), A11(indexes)];
    dlmwrite('../datafiles/dataset3real.tab', dataset,'delimiter','\t');

    % Clear data
    clearvars;
end
```

analysis.m

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <algorithm>    /* Sort n log(n) */
#include <string>
#include <stdlib.h>
#include <math.h>
#include <limits>
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */
#include <list>

// CGAL imports
/* A model for Kernel that uses Cartesian
coordinates to represent the geometric objects. */
#include <CGAL/Cartesian.h>
#include <CGAL/bounding_box.h>
#include <CGAL/Circle_2.h>
#include <CGAL/Exact_rational.h>
#include <CGAL/squared_distance_2.h> //for 2D functions
#include <CGAL/enum.h>
#include <CGAL/Arr_circle_segment_traits_2.h> // CGAL circle arrangements
#include <CGAL/Arrangement_2.h>

using namespace std;


typedef CGAL::Cartesian<CGAL::Exact_rational>          Kernel;
typedef Kernel::Circle_2                               Circle_2;
typedef Kernel::Point_2                                Point_2;

// Arrangements
typedef CGAL::Arr_circle_segment_traits_2<Kernel>      Traits_2;
typedef Traits_2::Curve_2                              Curve_2;
typedef Traits_2::CoordNT                              CoordNT;
typedef CGAL::Arrangement_2<Traits_2>                  Arrangement_2;

// Error msg
void errorMsg() {
    cerr << "Error!" << endl;
}
```

```
// PARSE METHODS
// Parse char* to int
void parseIntFromArgv(char* param, int& dest) {
    istringstream ss(param);
    if(!(ss >> dest)) errorMsg();
}

// Parse char* to double
void parseDoubleFromArgv(char* param, double& dest) {
    istringstream ss(param);
    if(!(ss >> dest)) errorMsg();
}

// Parse .tab file to vector<Point_2>
void readFromDataset(char* filename, vector<Point_2>& points) {
    int temp1, temp2;
    string input;
    ifstream fin;
    bool error = false;

    char filePath[120];
    strcpy(filePath, "../datafiles/");
    strcat(filePath, filename);
    strcat(filePath, ".tab");

    fin.open(filePath);

    while(true) {
        getline(fin,input);
        if(!fin) break; //check for eof

        istringstream buffer(input);
        buffer >> temp1 >> temp2;

        //check for non numerical input
        //and less/more input than needed
        if(!buffer || !buffer.eof()) {
            error = true;
            break;
        }

        points.push_back(Point_2(temp1, temp2));
    }
```

```
    if(error)
        cerr << "file is corrupted..." << endl;
}

// AUX METHODS
// Get boundaries (x y) of data
void getBoundaries(const vector<Point_2>& points, int& startX, int& endX,
        int& startY, int& endY) {
    Kernel::Iso_rectangle_2 r = CGAL::bounding_box(points.begin(), points.end());
    startX = CGAL::to_double(r.xmin());
    endX = CGAL::to_double(r.xmax());
    startY = CGAL::to_double(r.ymin());
    endY = CGAL::to_double(r.ymax());
}

// Generate circles by grid method
void generateCirclesByGrid(int startX, int startY, int rows, int columns,
        double cellSize, double bandwidthE2, vector<Circle_2>& circles) {
    double midCell = cellSize/2.0;
    double x = startX - midCell;
    double y = startY - midCell;
    CGAL::Exact_rational sqr_r1 = CGAL::Exact_rational(bandwidthE2);
    for(int i = 0; i < rows; ++i) {
        for(int j = 0; j < columns; ++j) {
            Point_2 center = Point_2(x, y);
            if(sqr_r1 == 0 || sqr_r1*1.0001 == 0 || sqr_r1*0.9999 == 0) circles.push_back(Circ
            else circles.push_back(Circle_2(center, sqr_r1, CGAL::CLOCKWISE));
            x = x + cellSize;
        }
        x = startX - midCell;
        y = y + cellSize;
    }
}

// Generate circles by kernel method
void generateCirclesByKernel(double bandwidthE2, const vector<Point_2>& points,
        vector<Circle_2>& circles) {
    for(int i = 0; i < points.size(); ++i) {
        Point_2 center = Point_2(points[i].x(), points[i].y());
        CGAL::Exact_rational sqr_r1 = CGAL::Exact_rational(bandwidthE2);
        circles.push_back(Circle_2(center, sqr_r1, CGAL::CLOCKWISE));
    }
}
```

```
// ALGORITHMS
// Number of combinations (N choose K)
uint nChoosek(uint n, uint k) {
    if (k > n) return 0;
    if (k * 2 > n) return k = n-k;  //remove the commented section
    if (k == 0) return 1;

    int result = n;
    for(int i = 2; i <= k; ++i) {
        result *= (n-i+1);
        result /= i;
    }
    return result;
}

// Get total intersections of 2d arrangement
void getTotalIntersections(const vector<Circle_2>& circles) {
    list<Curve_2> curves;
    Arrangement_2 arr;
    for(int i = 0; i < circles.size(); ++i) curves.push_back(Curve_2(circles[i]));
    insert(arr, curves.begin(), curves.end());

    uint totalIntersections = (uint)arr.number_of_vertices() - 2*circles.size();
    // Total intersections
    cout << totalIntersections << endl;

    // Locate the maximal degree.
    Arrangement_2::Vertex_const_iterator vit;
    std::size_t max_degree = 0;
    for(vit = arr.vertices_begin(); vit != arr.vertices_end(); ++vit) {
        if (vit->degree() > max_degree) max_degree = vit->degree();
    }
    cout << max_degree << endl;
}

// Circles arrangement
void getCirclesIntersectionsGrid(const vector<Circle_2>& circles, double bandwidth,
        double cellSize, int rows, int columns) {
    uint totalIntersections = 0;
    uint n = ceil(bandwidth/(cellSize/2))*4;
    uint intersections_approximation = nChoosek(n, 2)*rows*columns;

    // if(intersections_approximation < 500000)
    getTotalIntersections(circles);
    // else cout << 0 << endl << 0 << endl;
```

```
}

void getCirclesIntersectionsKernel(int nPoints, const vector<Circle_2>& circles,
        int startX, int endX, int startY, int endY) {

    double average_density_m_points = ((endX-startX) * (endY-startY))/nPoints;
    // if(average_density_m_points < 500)
    getTotalIntersections(circles);
    // else cout << 0 << endl << 0 << endl;
}

// Get incidences graph size
void createIncidenceGraph(const vector<Point_2>& points,
        const vector<Circle_2>& circles) {

    int sum_of_elems = 0;
    for(int i = 0; i < points.size(); ++i) {
        for(int j = 0; j< circles.size(); ++j) {
            if(circles[j].has_on_bounded_side(points[i]) ||
                 circles[j].has_on_boundary(points[i])) ++sum_of_elems;
        }
    }
    cout << sum_of_elems << endl;
}


int main(int argc,  char** argv) {
    if(argc != 4) errorMsg();
    else {
        // Parse data
        int startX, endX, startY, endY;
        double cellSize, bandwidth;
        vector<Point_2> points;
        vector<Circle_2> circlesGrid;
        vector<Circle_2> circlesKernel;
        parseDoubleFromArgv(argv[1], cellSize);
        parseDoubleFromArgv(argv[2], bandwidth);
        readFromDataset(argv[3], points);

        // Ratio = Bandwidth/cellSize
        cout << bandwidth/cellSize << endl;

        // Get the bounding box of data
        getBoundaries(points, startX, endX, startY, endY);
        int rows = ceil((endX-startX)/cellSize)+1;
```

```
        int columns = ceil((endY-startY)/cellSize)+2;

        // Generate circles
        generateCirclesByGrid(startX, startY, rows, columns, cellSize,
                bandwidth*bandwidth, circlesGrid);
        generateCirclesByKernel(bandwidth*bandwidth, points, circlesKernel);

        // Get circle arrangements
        cout << circlesGrid.size() << endl;
        getCirclesIntersectionsGrid(circlesGrid, bandwidth, cellSize, rows, columns);
        cout << circlesKernel.size() << endl;
        getCirclesIntersectionsKernel(points.size(), circlesKernel, startX, endX,
        startY, endY);

        // Get incidence graph of P points an C circles
        createIncidenceGraph(points, circlesGrid);
        createIncidenceGraph(points, circlesKernel);
    }
}
```

compute.m

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <algorithm>    /* Sort n log(n) */
#include <string>
#include <stdlib.h>
#include <math.h>
#include <limits>
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */

/* A model for Kernel that uses Cartesian
coordinates to represent the geometric objects.*/
#include <CGAL/Cartesian.h>
#include <CGAL/bounding_box.h>
#include <CGAL/Circle_2.h>
#include <CGAL/Exact_rational.h>
#include <CGAL/squared_distance_2.h> //for 2D functions
#include <CGAL/enum.h>

using namespace std;


typedef CGAL::Cartesian<CGAL::Exact_rational>   Kernel;
typedef Kernel::Circle_2                        Circle_2;
typedef Kernel::Point_2                         Point_2;

// Error msg
void errorMsg() {
    cerr << "Error!" << endl;
}

// PARSE METHODS
// Parse char* to int
void parseIntFromArgv(char* param, int& dest) {
    istringstream ss(param);
    if(!(ss >> dest)) errorMsg();
}
```

```
// Parse char* to double
void parseDoubleFromArgv(char* param, double& dest) {
    istringstream ss(param);
    if(!(ss >> dest)) errorMsg();
}

// Parse .tab file to vector<Point_2>
void readFromDataset(char* filename, vector<Point_2>& points) {
    int temp1, temp2;
    string input;
    ifstream fin;
    bool error = false;

    char filePath[120];
    strcpy(filePath, "../datafiles/");
    strcat(filePath, filename);
    strcat(filePath, ".tab");

    fin.open(filePath);

    while(true) {
        getline(fin,input);
        if(!fin) break; //check for eof

        istringstream buffer(input);
        buffer >> temp1 >> temp2;

        //check for non numerical input
        //and less/more input than needed
        if(!buffer || !buffer.eof()) {
            error = true;
            break;
        }

        points.push_back(Point_2(temp1, temp2));
    }

    if(error)
        cerr << "file is corrupted..." << endl;
}
```

```
// AUX METHODS
// Get boundaries (x y) of data
void getBoundaries(const vector<Point_2>& points, int& startX, int& endX,
    int& startY, int& endY) {
    Kernel::Iso_rectangle_2 r = CGAL::bounding_box(points.begin(), points.end());
    startX = CGAL::to_double(r.xmin());
    endX = CGAL::to_double(r.xmax());
    startY = CGAL::to_double(r.ymin());
    endY = CGAL::to_double(r.ymax());
}

// Generate circles by grid method
void generateCirclesByGrid(int startX, int startY, int rows, int columns,
    double cellSize, double bandwidthE2, vector<Circle_2>& circles) {
    double midCell = cellSize/2.0;
    double x = startX - midCell;
    double y = startY - midCell;
    CGAL::Exact_rational sqr_r1 = CGAL::Exact_rational(bandwidthE2);
    for(int i = 0; i < rows; ++i) {
        for(int j = 0; j < columns; ++j) {
            Point_2 center = Point_2(x, y);
            circles.push_back(Circle_2(center, sqr_r1, CGAL::CLOCKWISE));
            x = x + cellSize;
        }
        x = startX - midCell;
        y = y + cellSize;
    }
}

// Generate circles by kernel method
void generateCirclesByKernel(double bandwidthE2, const vector<Point_2>& points,
    vector<Circle_2>& circles) {
    for(int i = 0; i < points.size(); ++i) {
        Point_2 center = Point_2(points[i].x(), points[i].y());
        CGAL::Exact_rational sqr_r1 = CGAL::Exact_rational(bandwidthE2);
        circles.push_back(Circle_2(center, sqr_r1, CGAL::CLOCKWISE));
    }
}
```

```
// Count points of each circle generated
void countPointCircles(const vector<Point_2>& points,
    const vector<Circle_2>& circles, vector<int>& count) {
    for(int i = 0; i < points.size(); ++i) {
        for(int j = 0; j< circles.size(); ++j) {
            if(circles[j].has_on_bounded_side(points[i]) ||
                circles[j].has_on_boundary(points[i])) count[j] = count[j] + 1;
        }
    }
}

// ALGORITHMS
// Brute force problem 1
double bruteForce(const vector<Point_2>& points,
    const vector<Circle_2>& circles, double bandwidth) {
    double dist = sqrt(CGAL::to_double(CGAL::squared_distance(points[0], circles[0].center())))
    double result = abs(bandwidth - dist);
    double aux = 0.0;
    for(int i = 0; i < points.size(); ++i) {
        for(int j = 0; j < circles.size(); ++j) {
            dist = sqrt(CGAL::to_double(CGAL::squared_distance(points[i], circles[j].center())
            aux = abs(bandwidth - dist);
            if(aux < result) result = aux;
        }
    }
    return result;
}

// Brute force problem 2
double bruteForceDelta(const vector<Point_2>& points,
    const vector<Circle_2>& circles, double bandwidth, double delta) {
    vector<double> distances(points.size(), 0);
    double distance = numeric_limits<double>::max();
    int position = floor(delta*points.size());
    for(int i = 0; i < circles.size(); ++i) {
        for(int j = 0; j < points.size(); ++j) {
            distances[j] = abs(bandwidth -
                sqrt(CGAL::to_double(CGAL::squared_distance(circles[i].center(), points[j]))))
        }
        sort(distances.begin(), distances.end());
        if(distances[position] < distance) distance = distances[position];
    }
    return distance;
}
```

```
// TEST METHODS
// Geomask function
void geomask(const vector<Point_2>& points, double radius, vector<Point_2>& newPoints) {
    double angle, x, y, radius2;
    angle = x = y = radius2 = 0.0;
    srand (time(NULL));
    for(int i = 0; i < points.size() - 1; ++i) {
        radius2 = rand() / (RAND_MAX/radius);
        angle = ((rand() %360)*M_PI)/180.0;
        x = CGAL::to_double(points[i].x()) + cos(angle)*radius2;
        y = CGAL::to_double(points[i].y()) + sin(angle)*radius2;
        newPoints.push_back(Point_2(x, y));
    }
    radius2 = rand() / (RAND_MAX/radius);
    angle = ((rand() %360)*M_PI)/180.0;
    x = CGAL::to_double(points[points.size() - 1].x()) + cos(angle)*radius2;
    y = CGAL::to_double(points[points.size() - 1].y()) + sin(angle)*radius2;
    newPoints.push_back(Point_2(x, y));
}

int main(int argc, char** argv) {
    if(argc != 7) errorMsg();
    else {
        // Parse data
        int startX, endX, startY, endY, iterations;
        double initDelta, endDelta, cellSize, bandwidth;
        vector<Point_2> points;
        vector<Circle_2> circlesGrid;
        vector<Circle_2> circlesKernel;
        parseDoubleFromArgv(argv[1], cellSize);
        parseDoubleFromArgv(argv[2], bandwidth);
        readFromDataset(argv[3], points);
        parseDoubleFromArgv(argv[4], initDelta);
        parseDoubleFromArgv(argv[5], endDelta);
        parseIntFromArgv(argv[6], iterations);

        // Get the bounding box of data
        getBoundaries(points, startX, endX, startY, endY);
        int rows = ceil((endX-startX)/cellSize)+1;
        int columns = ceil((endY-startY)/cellSize)+2;
```

```
        // Generate circles
        generateCirclesByGrid(startX, startY, rows, columns, cellSize,
            bandwidth*bandwidth, circlesGrid);
        generateCirclesByKernel(bandwidth*bandwidth, points, circlesKernel);


        double r2Grid, r2Kernel;
        // Get thresholds
        if(iterations == 1) {
            r2Grid = bruteForceDelta(points, circlesGrid, bandwidth, endDelta/100.0);
            r2Kernel = bruteForceDelta(points, circlesKernel, bandwidth, endDelta/100.0);
            cout << r2Grid << endl;
            cout << r2Kernel << endl;
        }
        else if(iterations > 0) {
            double h = (endDelta - initDelta) / (iterations - 1);
            double val = initDelta;
            while(val <= endDelta ||
            (val < (endDelta*1.0001) && val > (endDelta*0.9999))) {
                r2Grid = bruteForceDelta(points, circlesGrid,
                bandwidth, val/100.0);
                r2Kernel = bruteForceDelta(points, circlesKernel,
                bandwidth, val/100.0);
                cout << r2Grid << endl;
                cout << r2Kernel << endl;
                val += h;
            }
        }

        vector<Point_2> newPointsByGridDelta;
        geomask(points, r2Grid, newPointsByGridDelta);
        vector<Point_2> newPointsByKernelDelta;
        geomask(points, r2Kernel, newPointsByKernelDelta);

        // Test results grid
        vector<int> countPointsInitialByGrid(circlesGrid.size(), 0);
        countPointCircles(points, circlesGrid, countPointsInitialByGrid);
        vector<int> countPointsGeomaskedByGridDelta(circlesGrid.size(), 0);
        countPointCircles(newPointsByGridDelta, circlesGrid,
         countPointsGeomaskedByGridDelta);
```

```
        // Test results kernel
        vector<int> countPointsInitialByKernel(circlesKernel.size(), 0);
        countPointCircles(points, circlesKernel, countPointsInitialByKernel);
        vector<int> countPointsGeomaskedByKernelDelta(circlesKernel.size(), 0);
        countPointCircles(newPointsByKernelDelta, circlesKernel,
         countPointsGeomaskedByKernelDelta);

                bool test = true;
        double maxDesviation = points.size()*delta;
        for(int i = 0; i < circlesGrid.size() and test; ++i) {
            double desviationGrid = abs(countPointsInitialByGrid[i] -
                countPointsGeomaskedByGridDelta[i]);
            if(desviationGrid > maxDesviation) test = false;
        }
        for(int i = 0; i < circlesKernel.size() and test; ++i) {
            double desviationKernel = abs(countPointsInitialByKernel[i] -
                countPointsGeomaskedByKernelDelta[i]);
            if(desviationKernel > maxDesviation) test = false;
        }

        if(test) cout << 1;
        else cout << 0;
        cout << endl;
    }
}
```

Referencias

[1] M. Löffler, J. Luo, R. I. Silveira. *Geomasking through Perturbation, or Counting Points in Circles*, In Abstracts 33rd European Workshop on Computational Geometry (EuroCG), pages 209-212, 2017.

[2] Paul A. Zandbergen, *Ensuring Confidentiality of Geocoded Health Data: Assessing Geographic Masking Strategies for Individual-Level Data*, Advances in Medicine, vol. 2014, Article ID 567049, 14 pages, 2014. doi:10.1155/2014/567049

[3] B. W. Silverman. *Density Estimation for Statistics and Data Analysis.* Chapman & Hall/CRC, April 1986.

[4] Hall, P., Sheather, S., Jones, M., & Marron, J. (1991). *On Optimal Data-Based Bandwidth Selection in Kernel Density Estimation.* Biometrika, 78(2), 263-269. doi:10.2307/2337251

[5] M. Kwan, I. Casas, and B. C. Schmitz. *Protection of geoprivacy and accuracy of spatial information: How effective are geographical masks?* Cartographica, 39(2):15–28, 2004.

[6] X. Shi, J. Alford-Teaster, and T. Onega. *Kernel density estimation with geographically masked points.* In Proc. Geoinformatics 2009, pages 1–4, 2009.

[7] Departament de Matemàrica Aplicada II. *Computational Geometry-GEI.* [en linea].
- [Consultado: 3 setiembre 2017]. Disponible en Internet:
<http://dccg.upc.edu/people/vera/teaching/courses/computational-geometry/>

[8] Fakultät für Mathematik und Informatik, *Voronoi Diagrams, chapters 1 and 2.* [en linea].
- [Consultado: 12 setiembre 2017]. Disponible en Internet:
<http://www.pi6.fernuni-hagen.de/downloads/publ/tr198.pdf>

[9] P. K. Agarwal and J. Matousek. *On range searching with semialgebraic sets. Discrete Comput. Geom*, 11:393–418, 1994.

[10] Donald Bren School of Information and Computer Sciences, *Nearest Neighbors and Voronoi Diagrams.* [en linea].
- [Consultado: 23 setiembre 2017]. Disponible en Internet:
<http://www.ics.uci.edu/~eppstein/junkyard/nn.html>

[11] CGAL website, *The Computational Geometry Algorithms Library.* [en linea].
- [Consultado: 5 octubre 2017]. Disponible en Internet:
https://www.cgal.org/

[12] Scrum website, *A Better Way Of Building Products*. [en linea].
- [Consultado: 30 setiembre 2017]. Disponible en Internet:
<https://www.scrum.org/resources/what-is-scrum>

[13] Matlab Website, *Matlab, el lenguaje del cálculo técnico*. [en linea].
- [Consultado: 18 octubre 2017]. Disponible en Internet:
<https://es.mathworks.com/products/matlab.html>

[14] D3 js website, *D3, Data-Driven Documents*. [en linea].
- [Consultado: 20 octubre 2017]. Disponible en Internet:
<https://d3js.org/>

[15] Plot.ly, charting library.
- [Consultado: 19 octubre 2017]. Disponible en Internet:
<https://plot.ly/javascript/>

[16] UK Data Service, *UK Data Service*. [en linea].
- [Consultado: 20 octubre 2017]. Disponible en Internet:
<https://www.ukdataservice.ac.uk/>

[17] The National Grid. [en linea].
- [Consultado: 18 diciembre 2017]. Disponible en Internet:
<https://www.ordnancesurvey.co.uk/resources/
maps-and-geographic-resources/the-national-grid.html>

[18] Gantter, *Gantter for Google Drive*. [en linea].
- [Consultado: 30 setiembre 2017]. Disponible en Internet:
<https://www.smartapp.com/gantterforgoogledrive>

[19] *TexMaker*. [en linea].
- [Consultado: 1 setiembre 2017]. Disponible en Internet:
<http://www.xm1math.net/texmaker/>

[20] Sublime Text. [en linea].
- [Consultado: 1 setiembre 2017]. Disponible en Internet:
<https://www.sublimetext.com/>

[21] Sublime Text Package Control. [en linea].
- [Consultado: 16 octubre 2017]. Disponible en Internet:
<https://packagecontrol.io/>

[22] A successful Git branching model. [en linea].
- [Consultado: 16 octubre 2017]. Disponible en Internet:
<http://nvie.com/posts/a-successful-git-branching-model/>

[23] Kernel distribution. [en linea].
- [Consultado: 22 octubre 2017]. Disponible en Internet:
<https://es.mathworks.com/help/stats/kernel-distribution.html>

[24] Chainey, Spencer. *Examining the influence of cell size and bandwidth size on kernel density estimation crime hotspot maps for predicting spatial patterns of crime.*, Bulletin of the Geographical Society of Liege, 2013.

[25] Becky P. Y. Loo and Tessa Kate Anderson. *Spatial Analysis Methods of Road Traffic Collisions (1st ed.)*. CRC Press, Inc., Boca Raton, FL, USA, 2015.

[26] Kernel smoothing function estimate for univariate and bivariate data. [en linea].
- [Consultado: 22 octubre 2017]. Disponible en Internet:
<https://es.mathworks.com/help/stats/ksdensity.html>

[27] CGAL 4.11 2D Arrangements [en linea].
- [Consultado: 10 diciembre 2017]. Disponible en Internet:
<https://doc.cgal.org/latest/Arrangement_on_surface_2/index.html>

[28] Clustering datasets. [en linea].
- [Consultado: 10 diciembre 2017]. Disponible en Internet:
<https://doc.cgal.org/latest/Arrangement_on_surface_2/
Arrangement_on_surface_2_2circles_8cpp-example.html>

[29] Fogel, Efi; Halperin, Dan; Wein, Ron. *CGAL Arrangements and Their Applications A Step-by-Step Guide. Geometry and Computing*, doi:10.1007/978-3-642-17283-0, Springer-Verlag Berlin Heidelberg, 2012.

[30] Two-sample Kolmogorov-Smirnov test. [en linea].
- [Consultado: 28 diciembre 2017]. Disponible en Internet:
<https://es.mathworks.com/help/stats/kstest2.html>

[31] Beware the Kolmogorov-Smirnov test [en linea].
- [Consultado: 28 diciembre 2017]. Disponible en Internet:
<https://asaip.psu.edu/Articles/beware-the-kolmogorov-smirnov-test>

[32] Median-Finding Algorithm. [en linea].
- [Consultado: 18 diciembre 2017]. Disponible en Internet:
<https://brilliant.org/wiki/median-finding-algorithm/>

[33] Clustering datasets. [en linea].
- [Consultado: 23 diciembre 2017]. Disponible en Internet:
<http://cs.uef.fi/sipu/datasets/>